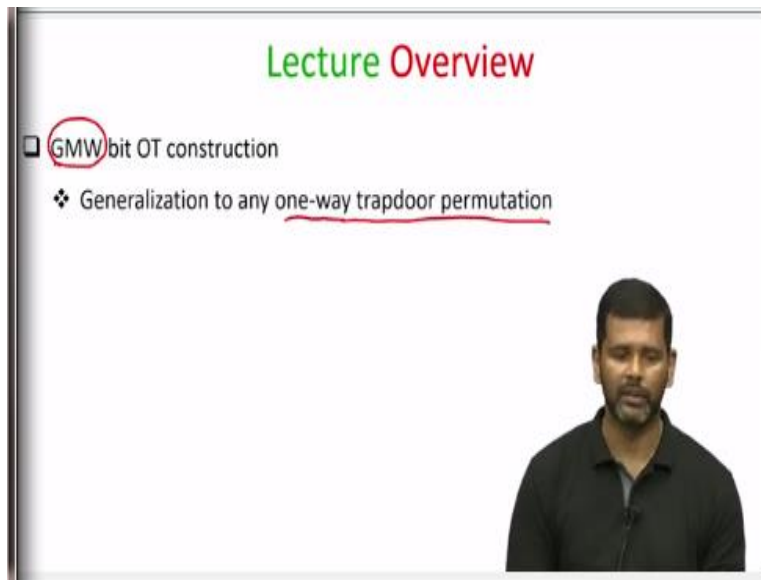


Secure Computation: Part I
Prof. Ashish Choudhury
Department of Computer Science Engineering
Indian Institute of Technology-Bengaluru

Lecture-39
Bit OT Based on RSA Assumption and Hard-Core Predicate

Hello everyone, welcome to this lecture. So, in this lecture we will see the construction of bit OT based on RSA assumption.

(Refer Slide Time: 00:37)

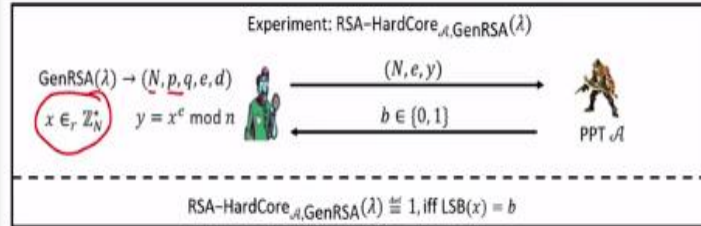


And this is basically the OT construction given in the GMW paper itself. And then we will see that this specific instantiation of OT based on the RSA assumption and RSA hardcore predicate, how it can be generalized to an OT construction based on what we call as one way trapdoor permutations.

(Refer Slide Time: 01:04)

RSA Hard-Core Bit

Intuition: Given the output of the RSA function on a random input, hard to compute input's LSB



Definition (RSA Hard-Core Predicate): For every PPT \mathcal{A} , there is a negligible function $\text{negl}(\lambda)$:

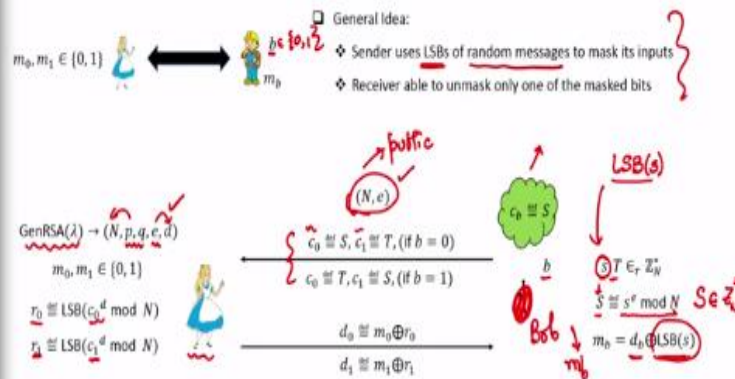
$$\Pr[\text{RSA-HardCore}_{\mathcal{A}, \text{GenRSA}}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$$

So, this is quickly the recap of RSA hardcore bit, the intuition here is that if there is an x which has been randomly chosen. And if someone gives you the value of $x^e \bmod N$ but does not tell you x , where N is the product of 2 random prime numbers p and q , then computing the LSB of x is difficult, you cannot do anything better than probability half plus negligible.

That means in no polynomial time algorithm can do better than just guessing the LSB of x . That means computing the LSB of x is indeed a difficult problem. So, that is the RSA hardcore bit or RSA hardcore function.

(Refer Slide Time: 01:49)

1-out-of-2 Bit-OT Based on RSA Assumption



Based on that let us see the construction of a very simple 1 out of 2 bit OT because here, sender has a pair of bits, m_0, m_1 , which is the private input of the sender. And the receiver Bob has a choice bit b which is Bob's private input. We want to design a protocol according to which Alice and Bob should interact and in which finally Bob should receive only the message he is interested in. So, if $b = 0$, he should only receive m_0 , if $b = 1$ he should only learn m_1 .

But in the process, Bob should not learn anything about the other message, namely the message with index $1 - b$. And other way around we want the following, that if Alice is bad and when she's interacting with Bob in the protocol, she should not learn which of the 2 messages m_0 or m_1 got transferred to Bob. That means she should not run what exactly was Bob's choice bit b .

And the general idea of the protocol will be the following. Alice, who is the sender here she will be using LSB's of some random messages to mask her inputs m_0 and m_1 . So, for m_0 she will be picking LSB of some random message. And for masking m_1 she will be picking LSB of another random message. And communicate the masked version of m_0 and m_1 to Bob.

And somehow Bob will have a mechanism to unmask only the message with index b , he will not be given sufficient resources or sufficient information to unmask the other message. That is the intuition behind this protocol. So, now let us see how exactly we instantiate this intuition? We do not know why this happened, so this is Bob here, we should have a picture of Bob here, I do not know why this has happened here, some rendering issue here.

So, Alice and Bob start the protocol as follows. So, Alice will have the pair of inputs m_0 and m_1 , that is our secret input. And Bob starts the protocol with his choice bit, which is a secret input. Now what Alice does is the following, she runs the RSA parameter generation algorithm. As part of that she will be picking random prime numbers p and q , which are distinct of λ bit size each, and compute the modulus by taking their product.

She will then pick a random e which is co-prime to $\phi(N)$ and with respect to that, run the extended Euclidean algorithm and compute d and now she makes an N and e public. So, you can imagine that every time Alice, you can imagine that for all the OT instances for which Alice and Bob are

going to interact, Alice can run this GenRSA function, generate the parameter and make N and e public once for all.

And after that the same setup can be used for polynomially many number of OT instances. Now Bob has this secret bit b , so, what it does is the following. It picks 2 random messages from the set \mathbb{Z}_N^* - s and T - he knows both those messages. And now he computes the output of the RSA function on the input s . The output of the RSA function of s will be $s^e \bmod N$.

And this he can compute, why? Because Bob has the knowledge of e , Bob has the knowledge of N , so he can compute $s^e \bmod N$. And as a result, he will obtain an element of \mathbb{Z}_N^* again because remember the RSA function is a permutation or a bijection from \mathbb{Z}_N^* to \mathbb{Z}_N^* . So, he will obtain an element, I denote it by S which is an element of \mathbb{Z}_N^* .

So, now he has 2 elements, one element is T another element S . Now, what he does is the following. He has to somehow send the elements T and S , but he is going to send in a shuffled order depending upon what exactly is his choice bit. If his choice bit is $b = 0$ then he will send the elements S and T in the following order, he will send S followed by T . So, I am denoting them as c_0 and c_1 , whereas if his choice bit is 1, then he sends T followed by S .

Now, Alice is completely unaware about the ordering, she knows that Bob is going to send her 2 random elements from \mathbb{Z}_N^* . By the way, this element S is also a random element from \mathbb{Z}_N^* because the element s was a random element from \mathbb{Z}_N^* . And mapping $x^e \bmod N$ is a bijection. So, Alice knows that Bob is going to send her 2 random elements from the set \mathbb{Z}_N^* .

But whether it is element S followed by element T or whether it is the element T followed by element S , she does not know, because both of them are random group elements. And the order in which Bob has sent the elements S and T depends upon that choice bit of Bob. So, basically depending upon what b is we can say that Bob knows that the element $c_b = S$. Now, the thing to note here is that Bob knows the LSB of the element s because he himself has picked it.

So, this is known to Bob that is a crucial piece of information you should remember. That means Bob knows that, so we will come back to this fact that how exactly will the knowledge of this LSB of the element s play a crucial role? Now, what should Alice do? Remember, our intuition was she has to use the LSB's of some random messages to mask her inputs m_0 and m_1 . So, that later it should be possible for Bob only to unmask one of the messages but not possible to unmask the other message.

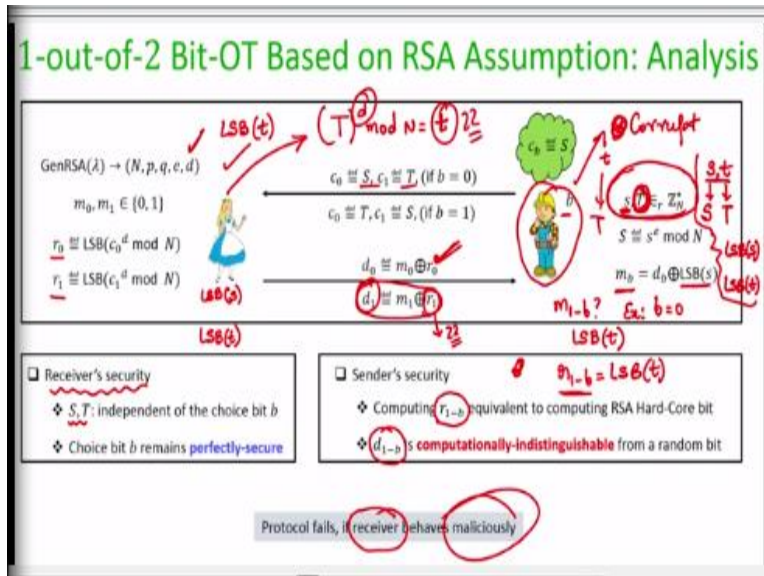
So, here is how Alice is going to mask her input, so she is computing the masks as follows. Irrespective of the order in which she receives the 2 elements, it could be S followed by T or it could be T followed by S , she calls the first element as c_0 and the second element as c_1 . She computes c_0^d that will give her an element of \mathbb{Z}_N^* and she can do that, because she has the exponent d with her.

So, she can compute $c_0^d \bmod N$ and take the LSB of the resultant element - that will be r_0 . And she takes the element c_1 ; compute $c_1^d \bmod N$ and takes the LSB of that element call it as r_1 . Now, what has happened here? The thing is one of these 2 elements c_0 and c_1 is definitely S . And when she computes $S^d \bmod N$, that will give Alice the element s .

And LSB of s will be known both to Alice and Bob. And it is LSB of s which will be serving as the mask for the message m_b , which Bob is interested to finally receive, that is the intuition here. So, what Alice is going to do is once she computes the mask r_0 and r_1 , she sends the masked version of m_0 and m_1 . So, m_0 is masked with r_0 , m_1 is masked with r_1 . Now, Bob is interested to receive the message m_b - that is what is the goal of Bob.

So, how he is going to receive m_b ? He will just take the message d_b , which Alice has sent to Bob. And XOR it back from the LSB of element is s and that will help him to cancel the effect of r_b . Because m_b was XORed with LSB of s and to get r_b , Bob just have to take the LSB of the element is s .

(Refer Slide Time: 11:32)



So, that is a very simple protocol, now let us do the analysis, why this protocol is going to maintain Alice privacy, and Bob's privacy and so on. So, let us first consider receiver's security. Namely, we consider the case when Alice is corrupt and try to ask whether she learns anything about b and the answer is no. In fact even if Alice is computationally unbounded, she cannot figure out what is b except with probability half.

She can always guess what is the value of b , we can never prevent that from happening. But even if she is computationally unbounded, she cannot do anything better than that, why? Because the elements S and T are picked completely independent of the element is b , it does not matter whether $b = 0$ or whether $b = 1$, Bob is just picking 2 random elements from the group and sending it, that is all.

Alice does not know the value of S , the value of T , so she is just seeing 2 random group elements and those 2 random group elements could be any 2 group elements independent of what exactly is the value of b is and that ensures Bob security or receiver's security. So, this OT protocol in fact gives us perfect security for the case when the sender is corrupt. Now, let us try to understand the case when now Bob could be correct.

Because during the OT protocol, it could be either Alice or Bob who could get corrupt. So, now if Bob is corrupt, definitely he will learn m_b because that is what he is interested in. But now we

have to argue whether he learns anything about the message m_{1-b} . And to find out other message namely the message with index $1 - b$, he has to find out the mask r_{1-b} . Because if he does not know the mask r_{1-b} , how can he find out?

Because he is receiving c_0 and c_1 both of them, say for instance b was 0. If $b = 0$, he can always compute this thing because r_0 is nothing but LSB of the element s which Bob himself has picked, so he can compute this part. But even though he is getting the value of c_1 , he will not be knowing what is the value of r_1 . Why he will not be knowing the value of r_1 ?

Because computing r_1 is equivalent to solving an instance of RSA hardcore function, why so? Because this element T , you can imagine that if I compute $T^d \bmod N$ suppose that gives us element is t . Alice can compute $T^d \bmod N$ because she has the decryption exponent d or she can invert the RSA function because she has the knowledge of d .

And hence she can compute the element t and hence she can compute the LSB of t , she can compute. But can Bob compute LSB of element t ? No, remember the mask r_0 and r_1 they are the LSBs of element s and t depending upon in what order the group elements have been communicated. So, r_0 and r_1 they are the LSBs of elements s and t .

Bob can compute one of those LSBs because he also has the element s , but he does not have the element t . Because he cannot invert the RSA function or he cannot solve the in instance of RSA hardcore function. So, basically Bob is facing here a challenge is to get the LSB of the element is t which is equivalent to solving an instance of RSA hardcore function and assuming that the RSA hardcore function is a difficult problem.

The LSB of the element is t which is nothing but the mask r_{1-b} , is like a random bit for the receiver Bob here. And if this mask r_{1-b} is almost like a random bit, then so is this other masked message, namely d_{1-b} . So, that means even though he is receiving the masking of the message m_{1-b} , the mask itself is computationally indistinguishable for this Bob because he cannot solve an instance of RSA hardcore function.

And hence he does not learn anything about the other message m_{1-b} . Now, an interesting thing here to notice the following - our protocol will fail to achieve sender's security, if the receiver behaves maliciously. So, our protocol will work under the assumption that Bob is indeed following the protocol instructions. That means even if it is under the control of the adversary, it behaves in a passive fashion or an eavesdropping fashion.

That means it does not deviate from the protocol instruction. But if Bob behaves maliciously, that means does not follow the protocol instructions then he can learn both the messages m_0 and m_1 , how? So, he is supposed to pick an element s and convert it into S but for the other message or the other group element, he is not supposed to learn the corresponding RSA inverse, instead what he can do is the following.

He can start with 2 random elements s and t , s is converted to S , a t is converted to T , how? By computing $s^e \bmod N$. And this t can be converted into T by computing $t^e \bmod N$. And now he will be knowing both the LSB of s as well as this LSB of t because he himself has picked it. Remember, this is a deviation from the protocol.

In the protocol he is not supposed to pick first elements the s and t and convert them into S and T , he is only supposed to pick one of the elements and compute output of the RSA function. For the other element, the difficulty for him should be to invert it but he is doing it other way around. Other way around in the sense he is basically picking actually 2 elements and computing root of the RSA function.

And then forwarding S and T , say as per the protocol. Now Alice will be using LSB of s for masking one of the message and LSB of t for masking another message. And both those things are known to a malicious Bob, now and he can unmask both d_0 as well as d_1 and learns both m_0 and m_1 . It is only when Bob picks one of the messages and computes the output of RSA function.

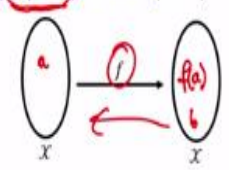
But for the other element he does not compute the inverse or not able to compute the inverse, the security of Alice is preserved in this protocol. So, that is why this protocol cannot tolerate a potential malicious behaviour from a Bob. Later on when I will offer the second part of this course,

we will see that how we can design OT protocols which can take care even against the potential malicious behaviour from Bob.

But this protocol will achieve security as long as Bob is honestly following the protocol instructions. Namely, for one of the elements he knows LSB but for the other element, he does not know the LSB. Remember, the LSB of t is not used for masking the message; T is inverted to get the element t which only Alice can do. But malicious Bob could do the following; he can always first take the small element t and convert it into T and then forward T and S in which case both the LSBs will be known to Bob.

(Refer Slide Time: 20:46)

One-Way Trapdoor Permutation (OWTP)

<p>❑ One-way trapdoor permutation (OWTP)</p>  <ul style="list-style-type: none"> ❖ Easy to compute for any input from the domain ❖ Difficult to invert any random input from codomain ❖ Easy to invert any input from the codomain, given the knowledge of the trapdoor information 	<p>❑ A trapdoor permutation scheme T over a finite set X is a triplet of algorithms (Gen, f, Inv)</p> <ul style="list-style-type: none"> ❖ $Gen() \rightarrow (pk, sk)$ <ul style="list-style-type: none"> ➢ pk: public key ➢ sk: secret key ❖ $f_{pk}: X \rightarrow X$ <ul style="list-style-type: none"> ➢ $f_{pk}(x) = y$ ➢ Deterministic algorithm ❖ $Inv_{sk}: X \rightarrow X$ <ul style="list-style-type: none"> ➢ $Inv_{sk}(y) = x$ ➢ Deterministic algorithm <p style="text-align: right; color: red; font-style: italic;">if sk is not known</p>
<p>❑ Correctness property ... for every possible outputs (pk, sk) of Gen and for all $x \in X$:</p> <p style="text-align: center; background-color: yellow;">$Inv_{sk}(f_{pk}(x)) = x$</p>	
<p>❑ One-Wayness: function f_{pk} should be a one-way function (OWF), even if an adversary knows the public key pk</p>	

So, now let us try to generalize this protocol based on what we call as one-way trapdoor permutation. So, for that let me quickly go through what is one-way trapdoor permutation? So, it is a one-way permutation from a set X to X , namely it is a bijection. And it is easy to compute for any input from the domain, that you give me any element a from the domain, I can always compute $f(a)$ in polynomial amount of time.

But if you give me an element b from the core domain and ask me to invert it in polynomial amount of time, then it is difficult, except with a negligible probability I cannot do that. However, if I consider one-way trapdoor permutation, then it is possible to invert the function provided you have

some special information available. And that special information is called as the trapdoor information.

So, that means a regular one-way permutation is we cannot invert it on random input from the co-domain. But if I am talking about one-way trapdoor permutation, then there is always some special trapdoor information associated with that permutation which can allow me to even invert the function on random inputs from the co-domain. So, let us formally define this one-way trapdoor permutation.

So, a one-way trapdoor permutation scheme \mathcal{T} is a triplet of algorithms. So, there will be a parameter generation algorithm which will generate a public parameter pk and a secret parameter sk in polynomial amount of time. There will be 2 functions, a function f_{pk} in the forward direction and an inverse function Inv_{sk} in the reverse direction. The forward direction function is operated by the public parameter and it is a deterministic function where, if you are given the public parameter and the input x , you can compute the output y in polynomial amount of time. The inverse function is operated by the secret parameter sk , where if you are given the description of the inverse function and the secret parameter then again in polynomial amount of time, you can invert or compute output of inverse function for any input y . And these 2 functions f_{pk} and Inv_{sk} are inverse of each other.

In the sense that we have this correctness property which says that for every pair of parameters pk, sk generated by the parameter generation algorithm. If you compute the output of the f_{pk} function on the input x and then later compute the inverse of that, you should get the same output. And we have this one-wayness property, which says that if you do not know the value of sk , then you cannot invert the function.

That means your function f_{pk} should behave like a one-way function if you are given only the public parameter but not the secret parameter. That means the output cannot be computed even if you know the description of the inverse function. That means until and unless you are not given the secret parameter, you cannot compute the output of that inverse function on any input of your choice that is the one-wayness property.

(Refer Slide Time: 24:39)

Goldreich-Levin (GL) Theorem

- ❑ Shows how to construct hard-core predicates for OWF/OWP
- ❑ Let f be a OWF/OWP
- ❑ Consider the following 2-input function g , constructed from f
- ❑ Claim : If f is a OWF/OWP then g is also a OWF/OWP

Given $(f(x), r)$
 \downarrow
Then $f(x)$
 \downarrow
 x

$g(x, r) \stackrel{\text{def}}{=} (f(x), r)$, where $|x| = |r|$

- ❑ **GL Theorem:** The following function is a hard-core predicate for function g :
 $gl(x, r) \stackrel{\text{def}}{=} r_1x_1 \oplus r_2x_2 \oplus \dots \oplus r_nx_n$ where $x = (x_1, \dots, x_n)$ and $r = (r_1, \dots, r_n)$
- ❑ Intuition:
 - ❖ Given $f(x)$ for a random x , computing XOR of a random subset of bits of x in polynomial time is hard

I give you $(f(x), r)$

Now we have a very nice result which says that if you are given a one-way function, one-way permutation then you can always associate a hardcore function with that one-way function or one-way permutation, which is often called as the Goldreich-Levin theorem. So, let f be a one-way function or one-way permutation, then we can always convert this function f to another function g which is a one-way function or one-way permutation, how we can convert this function f to the function g ?

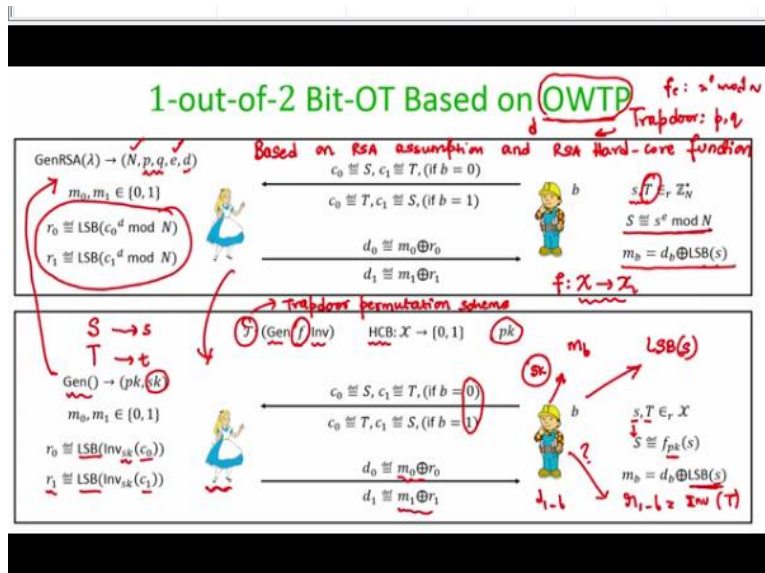
Well, g will be now a 2 input function even though your f was a 1 input function, but the other input of this g function is always produced in the output. And the first part of the output is the output of the function on the input x . So, basically you can imagine that g is almost the same as your f function, it is just that I am supplementing it with a random input or some other part of the input which will be given as it is in the other part of the output, where the length of the 2 inputs for the g function should be same.

And a claim is that if your function f was a one-way function or a one-way permutation and so is the function g . And this can be very easily proved through a reduction, namely, we can prove that if given just $f(x)$ and r , you can compute back x in polynomial amount of time. Then using the same algorithm, I can do the following, just given $f(x)$, I can compute x in polynomial amount of time.

But that is a contradiction to the assumption that f was a one-way function or one-way permutation at the first place. Now, the GL theorem is the following, it says that corresponding to this function g , the associated hardcore function is the following. So, the hardcore function associated with this function g is the following. It is basically equivalent to computing this XOR. Namely, computing a XOR of random subsets of bits of x , why random subsets of bits of x because r will be chosen uniformly at random.

So, basically it says that even if I give you $f(x)$ as well as r but I do not tell you what is x . Then computing this linear combination of the bits of x in polynomial amount of time with probability better than half is not possible, that is what this theorem proves. Again, all these things I am taking from my earlier course on foundations of cryptography, I am quickly recapping all those things.

(Refer Slide Time: 27:47)



Now, what we are going to see is the following. We have seen the 1 out of 2 bit OT based on the RSA hardcore function, so this was based on RSA assumption and RSA hardcore function, we will now generalize whatever actions Alice and Bob have taken this protocol based on we have a one-way trapdoor permutation and its associated hardcore function. By the way, this RSA permutation is indeed a one-way trapdoor permutation because the forward direction function was f_e and the trapdoor information there are the prime factors of your N namely p and q .

Because if someone has this trapdoor p and q then it can compute the exponent d and then it can invert y , namely it can compute $y^d \bmod N$. So, what we are now going to see is instead of RSA one-way trapdoor permutation if you are given any one-way trapdoor permutation and by the way, we have several candidate one-way trapdoor permutation available based on various number theoretic hard problems.

So, you take any one-way trapdoor permutation and follow the following blueprint which I am going to discuss and that will give you a 1 out of 2 bit OT. Why this is important, why studying the generalized version of the OT construction is important? Because if tomorrow there is any weakness which have which is reported with respect to the RSA permutation, then you can follow the blueprint that I am going to discuss and then replace the RSA one-way trapdoor permutation by any other candidate one-way trapdoor permutation against which no weaknesses are yet reported.

So, this will be the general OT protocol, here again Alice will have 2 messages which are bits. Bob has a selection bit. Now, Alice will do the following. So, we will assume that we have trapdoor permutation scheme available whose details are publicly known, trapdoor permutation scheme namely everyone knows the parameter generation algorithm the function f and its inverse and there is a corresponding hardcore bit associated here.

By the way, I am assuming that this function f is from the set \mathcal{X} to \mathcal{X} . Everyone knows all those details. Now, Alice is going to generate the parameters as follows. So, she will run the parameter generation algorithm which will give her a public parameter and a secret parameter and she is going to make the public parameter available in the public domain. Now, if you compare this step with the RSA based OT protocol, this step is nothing but Alice running the RSA parameter generation algorithm and making N and e available in the public domain.

And she is retaining this secret parameter sk , the secret parameters were p , q and d which were written by Alice in the RSA based OT protocol. Now, what Bob is going to do is Bob is going to pick an element s whose LSB he knows and another element T . This is equivalent to Alice Bob

picking the group elements s and T from \mathbb{Z}_N^* , but now I am not taking \mathbb{Z}_N^* because I am now taking about an abstract function f which is from the set \mathcal{X} to fancy \mathcal{X} .

What is going to do is he is going to compute the output of the one-way function on the input s . How he can compute well he knows the public parameter. So, he can always compute this and he knows the description of the function f . Again if you compare it with the RSA based OT this step is equivalent to Bob computing the output of RSA function on the group element s and now depending upon his choice bit whether it is $b = 0$ or 1 , he is going to send the elements S and T .

So, if it is 0 , then he has sent S followed by T , if it $b = 1$ then he sends T followed by S . The idea is that Bob is interested to receive m_b and this message m_b Alice should somehow encrypt using the LSB of the s which is known only to Bob. But Alice should not be aware of which of these 2 messages or which of these 2 elements is S and T is actually the output of f function on the element s because she is not aware of the element s .

So, what she is going to do is she is going to compute the inverse of both the elements S and T . By the way, she does not know whether c_0 is S or whether c_1 is S in whatever order she has received the elements to take them in the same order and just compute the inverse function of the trapdoor permutation. Can she compute? Yes, because she has the knowledge of the secret parameter and focus on the LSB of the recovered elements.

So, basically, one of these r_0 and r_1 is the LSB of the element s and other mask here is the LSB of the element t what exactly is the case Alice is completely unaware. And again, this step is not equivalent. It is generalization of the step wherein the RSA based OT protocol Alice has computed the RSA inverse for the element c_0 and c_1 . And now she is going to do the following.

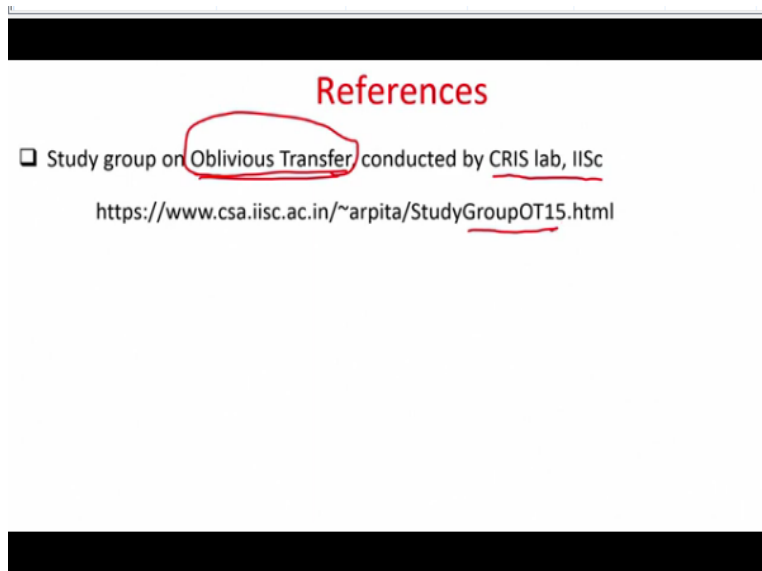
She will send a masking of the message m_0 , a padded version of the message m_0 , where r_0 is serving as the one time pad and she is padding m_1 with the message bit r_1 . Now Bob will be able to unmask 1 of these to receive the messages. So, what he can do is he can take the element as s which he himself has picked. Focus on its LSB and unmask it from the element d_b . For the other

element d_{1-b} , he cannot do anything because he would not be knowing what is the LSB or the pad r_{1-b} .

Because this is nothing but the inverse of the element T , but he cannot compute the inverse of the element big T because he do not have the knowledge of secret parameter and that exactly was the case for the RSA based OT protocol, Bob there could compute m_b , but he cannot compute f_{1-b} because the other message is padded or masked with r_{1-b} which is the LSB of the inverse of the element T which he cannot compute because that is equivalent to computing the hardcore function or hardcore bit.

So, now you can see that how this RSA based OT protocol can be generalized to an OT protocol where it could be any one-way trapdoor permutation and it is associated hardcore bit which is used to do the masking of the inputs of Alice.

(Refer Slide Time: 36:25)



So, a lot of things which I have used in today's lecture namely RSA assumption, hardcore bit etcetera those details I have taken from my earlier course on foundations of cryptography which is available on NPTEL. As part of oblivious transfer again, as I said, when we started discussing about oblivious transfer, it is a fundamental problem in cryptography protocols.

Here are plenty of resources, plenty of research, which is going on with respect to oblivious transfer. So, at Indian Institute of Science CRIS lab, very nice study group was conducted on oblivious transfer itself where they discussed lots of fundamental results related to oblivious transfer, also discussed state of the art related to oblivious transfer. So, a lot of materials which I have taken for preparing the slides are taken from this source. With that, I end this lecture. Thank you.