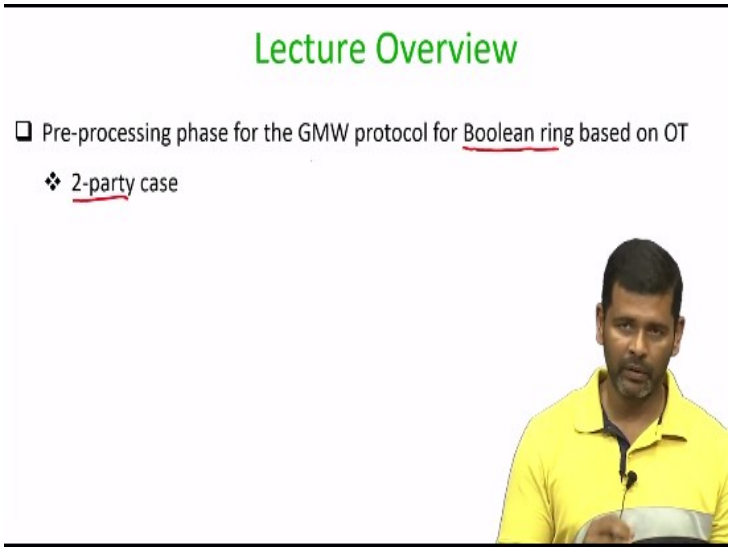
Secure Computation: Part 1
Prof. Ashish Choudhury
Department of Computer Science & Engineering
International Institute of Information Technology-Bengaluru

Lecture - 42
Pre-Processing Phase for the GMW Protocol

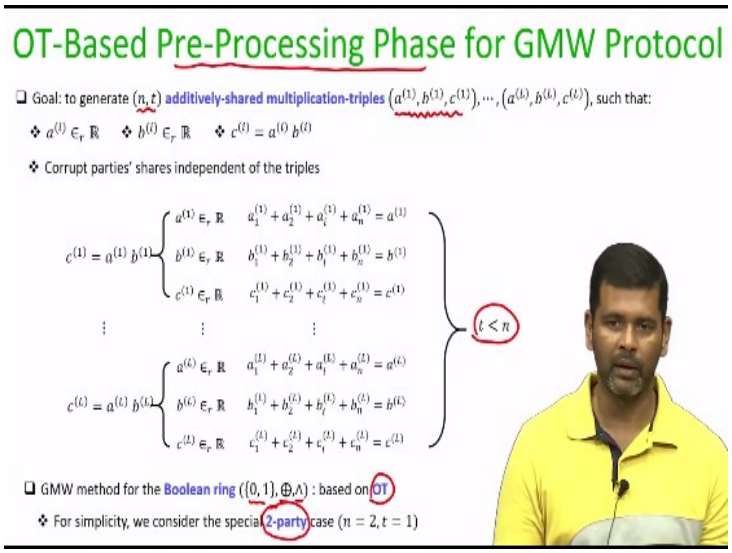
Hello everyone, welcome to this lecture. So now we have seen oblivious transfer protocol. So let us go back to the problem of preprocessing phase for the GMW protocol.

(Refer Slide Time: 00:41)



And we will see the preprocessing phase for the GMW protocol for the simpler case of Boolean ring based on oblivious transfer. And for simplicity we will focus on the 2-party case in this lecture. We will then see in the next lecture how this method can easily generalize for the n-party case.

(Refer Slide Time: 01:02)



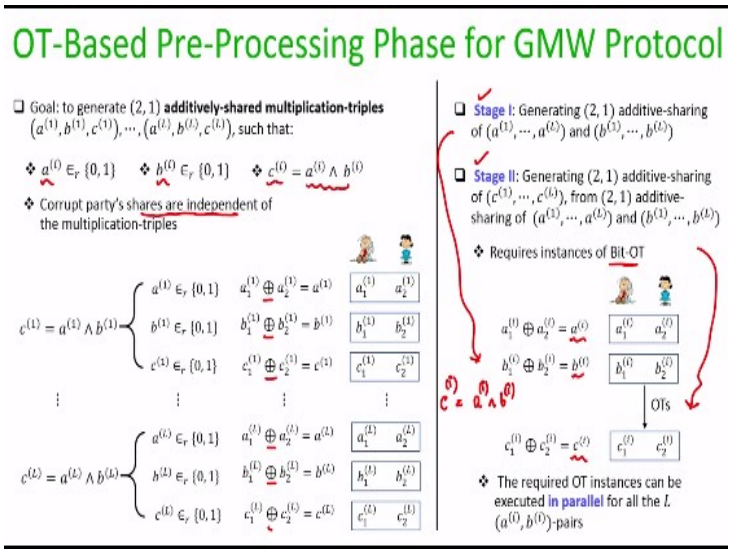
So just to recall what exactly we want to generate here. When we say the preprocessing phase our goal in the preprocessing phase is to generate additively shared random multiplication triples okay, where the a component, b component, c component they should be random elements from the ring except that the c component should be the product of the a and b component.

And all the components of all the triples should be n, t additively secret shared among the parties okay, where the addition operation is as per the underlying ring. And we want to do in the dishonest majority setting where all but up to n - 1 parties can be under the control of the adversary, okay.

And we will see that original GMW method here which was proposed for the special case where the ring R was the Boolean ring where we have only two elements namely 0 and 1. The plus operation is the XOR operation, the product operation or the multiplication operation is the and operation. And this is based on oblivious transfer.

And we will see first the simpler case of 2-party setting where there are 2 parties and 1 of them could be corrupt, okay.

(Refer Slide Time: 02:28)



So that means now if I rephrase the goal of the preprocessing phase for the 2-party case and for the Boolean ring, our goal is the following. We want to generate additively shared random multiplication triples and by additive sharing I mean that each party should have the share of the underlying value and there are two parties because I am talking about the 2-party case such that the XOR of their shares should give you the underlying value.

So the a component, b component c component they are going to be bits, okay. So the a components, b components, c components they are going to be bits. The a and b components are going to be random. The c component should be the and of the a and b components. And this data structure has to be generated in such a way that corrupt party's shares should be independent of what exactly are the underlying multiplication triples.

So generating this data structure will be done in two key stages. And first stage is for all the L multiplication triples which we want to generate. We generate first the a and b components in a secret shared fashion, okay. Now assuming that we have generated the a and b components in a secret shared fashion for all the L multiplication triples, we generate securely the c component, again in a secret shared fashion.

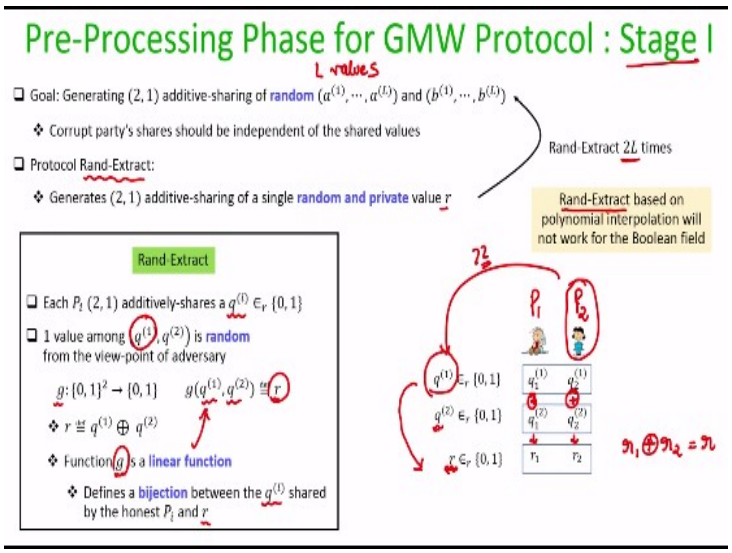
And generating the c components in a secret shared fashion requires instances of bit oblivious transfer. Namely if I take the ith multiplication triple, then in stage I we have generated this data structure. Namely, we have generated a secret shared bit a i, a

secret shared bit b i with none of the parties knowing the exact value of the bits a i and b i.

And now in stage II we have to use the oblivious transfer protocols in some ways so that parties hold secret sharing of the bit c i where c i is the and of a i and b i. And in this process, no additional information about the bits a i and b i should be revealed.

Now if we are doing this stage I and stage II computations in the preprocessing phase, the advantage is that the stage I as well as stage II for all the L multiplication triples that we want to generate can be executed in parallel because they are completely independent of each other.

(Refer Slide Time: 05:17)



So now let us see first the stage I and what is the goal of stage I? We want to generate the a and b components, the bits a and b for all the L multiplication triples in a secret shared fashion among the two parties with no party knowing the exact values of the a bits and b bits. And for this we will use the protocol which I call as Rand-Extract. We had seen a Rand-Extract protocol for the case of sum a secret shared b processing phase, okay.

So I have used a notation and called the protocol Rand-Extract for this GMW preprocessing as well. And this Rand-Extract protocol will help you to generate additive sharing of random values r okay, where none of the parties will be knowing

what exactly is the bit r. But the bit r will be secret shared as per the XOR sharing among the two parties.

And since our goal is to generate L such a and b random values, but this one instance of Rand-Extract can give you one such random value, we can run this Rand-Extract protocol 2L number of times because our goal is to generate L number of a bits randomly and L number of b random bits, okay. So we have 2L values to be generated.

And before going further, I would like to stress here that the Rand-Extract process that we had seen in the context of secret sharing based preprocessing phase it would not work for your Boolean ring or Boolean field because there we require the field, first of all it requires a finite field because it was based on polynomial interpolation and extending a polynomial to new points.

And that requires a field to be sufficiently large, okay. Whereas we are now talking about a method, the Rand-Extract method where we have only two values in my underlying ring. So that method of randomness extraction would not work for the Boolean ring setting. So let us see the randomness extraction method for the Boolean ring setting.

So each party P i here will pick a random value q i, which is going to be a bit because now we are considering the Boolean ring. And it will secret share it among the two parties. So what we are saying here is that P 1 picks the value of the bit q 1 and it secret shares it into two random bits. Secret shares returned two bits that means it picks two random shares q 1 1, q 1 2 such that their XOR gives the value q sub 1 and it acts as the dealer and gives the second share to the second party.

And in the same way the second party picks random bit q sub 2 acts as a dealer and secret shares it as per the additive secret sharing. And remember additive secret sharing here means the addition operation is the XOR operation. Now what we can say about the two bits q 1 and q 2 which are secret shared by these two parties. One of these two parties is not under the control of the adversary.

So suppose without loss of generality, it is the party P 1. That means the bit q 1 which has been secret shared by the party P 1 is random from the viewpoint of the party P 2. Because for q 1 it learns only a share. But what is the other share, she is not aware. And hence it could be any value q 1 which has been XOR shared by the first party.

So now to define the output of this randomness extraction process, we define a linear function. I call that function as g, which takes two bits and produces one bit as an output. And the output basically is the XOR of the bits shared by the respective parties. So if q 1 and q 2 are the bits respectively shared by the two parties, then r is defined to be the output, r is defined to be the XOR of those two input bits.

And if you see here, this function g is a linear function of the inputs. And now we know that this XOR sharing or additive secret sharing to be more specific, allows the parties to compute linear functions on the shares of the inputs. So what are the inputs for the function g here? The inputs are here q 1 and q 2.

None of the parties knows the exact means the P 1 does not know the full value of q 2. P 2 does not know the full value of q 1. So the parties hold only the shares of q 1 and q 2. So now if I ask the parties to apply the linear function on their shares of q 1 and q 2, they end up obtaining the shares of the output r.

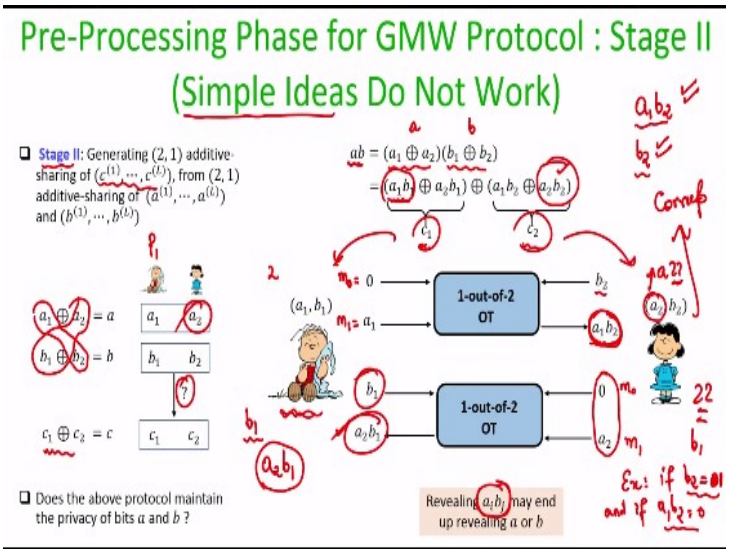
That means, if each party just locally performs an XOR of their shares of q 1 and q 2, that will help them to obtain shares of r value and now I can say that the XOR of r 1 and XOR of r 2 is nothing but r. That means, I have now generated a value i in a secret shared fashion. And now I can say that this value r is a random value from the viewpoint of this corrupt party P 2 okay, or whoever is the corrupt party.

This is because there is a well-defined bijection between the q i value among these two q values shared by the honest party and the output r. What does that mean? Well, since again for simplicity, I am taking P 1 to be the honest party, this q 1 value is a random value from the viewpoint of the corrupt P 2. It could be any bit. It could be 0 with probability half. It could be 1 with probability half.

And the way we have defined the output r as a function of q 1 and q 2, for every candidate q 1 which is corrupt P 2 things in her mind there is a candidate r. So if she thinks that okay q 1 is 0, then that value is 0 along with the q 2 shared by P 2 defines the r. Whereas if she makes the assumption that q 1 is 1 then that 1 value along XOR with q 2 defines the corresponding r.

But since q 1 is picked uniformly at random, it could be any q 1 from the viewpoint of P 2 which has been now secret shared and hence it could be any r which is now available among the two parties in a secret shared fashion. So that is a simple Rand-Extract protocol, which works for the Boolean ring case.

(Refer Slide Time: 12:14)



Now let us go to stage II, okay. That means assuming that the stage I has been executed 2L times and we have generated the bits a 1 up to a L and b 1 up to b L which are random and now they are available in a XOR shared fashion. Our goal is to now generate the corresponding XOR sharing of the corresponding c values. So let us do it for just one specific a and b pair.

The same method can be executed in parallel L times. So that means I am assuming that there is a bit a. No one knows the value of a. It is a uniformly random bit and it is XOR shared among the two parties. It has been generated by stage I. In the same way there is a uniformly random bit b which has been generated by running the protocol Rand-Extract and both the parties P 1 and P 2 have their respective bit shares for the bit b.

And now somehow they want to interact and obtain shares a bit shares c 1 and c 2 so that it XORs to c where c is the and of a and b. And in this process nothing about the exact value of a, exact value of b should be revealed, okay. That is what we want to ensure. So if we have this mechanism, this mechanism can be executed L times for all the a, b pairs which we have generated in the stage 1 and that will complete the preprocessing phase.

So let us first try to see a very simple idea which many of you might be tempting to propose, but the purpose of this slide is that we want to see that simple ideas do not often work, okay. So what is the and of a and b? So I am now not writing down the and operation; a times b in the Boolean ring context you should imagine it as a and operation, okay.

So a times b or the and of a and b is the and of these two expressions, because a is XOR of a 1 and a 2, b is XOR of b 1 and b 2. If I further expand and rearrange terms like this, let me say that the XOR of the first two expressions here, the first two summands, let it be c 1. And the XOR of the last two summands, let it be c 2. Now can this first party P 1, compute a 1 the value c 1?

Well, it can compute a 1 times b 1 because it has both a 1, it has b 1. But it cannot compute a 2 times b 1 because even though b 1 is with the first party a 2 is not available with the first party; a 2 is available with the second party. And we cannot afford to ask the second party to give a 2 in clear to the first party because if a 2 is given in clear to the first party, then that a 2 along with a 1 gives the value of a to the party P 1.

And that breaches the security requirement. We do not want to reveal anything additional about the a and b values here. In the same way party P 2, she has a 2 and b 2. So she can compute a 2 times b 2, fine. But what about a 1 times b 2? a 1 times b 2 is like computing a cross term here where one of the part is with first party, another part is with the second party.

And I cannot ask the first party to give a 1 in clear to the second party. Because that will now reveal the value of a to the second party. So even though I can identify here a partial c 1 and partial c 2 which somehow can be computed, will complete the secret sharing of a times b, there is some cross terms which are involved here, namely two cross terms.

So you can imagine here that there are two cross terms here which are the source of problem, the computation of a 1 times b 2 and the computation of a 2 times b 1. How do we compute it? So imagine we propose the following protocol. So since there are two cross terms here, we run two instances of OTs. In one instance of OT, P 1 becomes the sender and P 2 becomes the receiver.

And in another OT which can be executed in parallel symmetrically, P 2 can serve as the sender and P 1 can serve as the receiver. This is just to ensure that the balance of OT executions is maintained properly. Because if you see the OT protocol sender and receiver they perform asymmetric computation. Sender has a different kind of computations to be performed, receiver has different kinds of computations to be performed, right?

So you could have asked to P 1 to act as a sender in both the instances of OT here, but just to spread the load symmetrically, I am asking P 1 to act as the sender in one of the OTs and as a receiver in another OT instance. So now we have to decide what should be the messages which P 1 should feed as sender and P 2 should feed as receiver and so on.

So suppose in the first OT, we ask sender, we ask P 1 to act as a sender and set these as his messages. And we ask party P 2 the second party to set b 2 as her choice bit. So if you now see closely, the correctness property of the OT will ensure that b 2 receives this cross term a 1 times b 2 okay. And in this process if this first party who is the sender party if he is corrupt, he does not learn anything about b 2, okay.

Why? He does not learn anything about b 2 because that comes from the receiver's privacy because b 2 is the choice bit of the second party and indeed this OT protocol is a secure OT protocol. Then if the first party is the correct party, it does not learn

anything about the choice bit of the receiver who is the second party and hence it does not learn anything about b 2, fine.

And in the second instance of the OT we can ask the first party to select his choice bit to be 1 and let the second party acts as a sender with these inputs, m naught and m 1. As a result of the correctness of the OT protocol, the first party obtains the cross term a 2 times b 1 and a potentially corrupt b 2 does not learn anything about b 1, because b 1 is the choice of the receiver here.

And if indeed the OT protocol has receiver's privacy, then the second party who is acting as the sender does not learn anything about the choice bit which is b 1. And now you can see P 1 has the first party has everything. He anyhow has the ability to compute a 1 times b 1 and he has obtained a 2 times b 1 as a receiver of the OT, so he can compute c 1.

And in the same way, the second party she has the ability to compute a 2 times b 2 on her own. And as a receiver she has obtained a 1 times b 2 as the output of the first OT. So now if she XORs them, she gets c 2. So c 1 is now computable by the first party, c 2 is computable by the second party and now you might say that okay our stage II task is done. But now let us try to carefully analyze this protocol.

Does the protocol maintain the privacy of the bits of a and b? And the answer is no, because we are revealing a i times b j in clear to one of the receiving parties or one of the parties who is actually acting as the receiver and that may end up revealing the entire a or entire b. So imagine again a corrupt P 2. So if she is acting as a receiver, then she does not learn anything about b 1, that is fine.

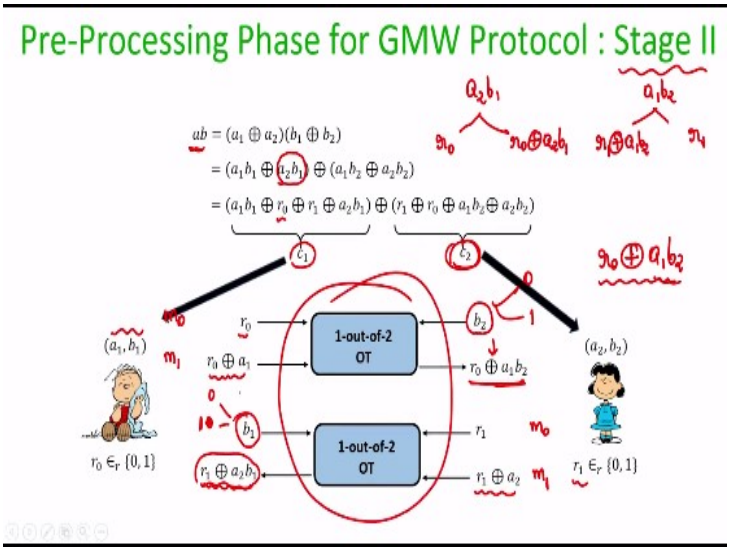
That comes from the security of receiver from the second OT. But as a receiver, she is learning something about a 1 here. Why? Because she is learning a 1 times b 2 in clear. She is learning this full value. And b 2 is also known to her. Now comparing a 1 times b 2 and b 2, she may end up learning a 1. How? Say for instance, if b 2 is equal to 0, sorry, if b 2 is equal to 1 say, and if a 1 times b 2 turns out to be 0, then she learns that a 1 is 0.

Because if a 1 would have been 1 and in anyhow b 2 is equal to 1, then she should have got a 1 times b 2 to be 1. But since she is getting a 1 times b 2 to be 0, and her own share b 2 is 1, she learns that a 1 is 0. And if a 1 is learned by the second corrupt party she is anyhow having the second share of a and that along with the first share of a helps her to completely find out what is the value of a, okay.

In the same way, if say the sender would have been corrupt, sender sorry, if the first party would have been corrupt, he learns a 2 times b 1 in this OT instance as a receiver. Now he learns a 2 times b 1, he also have b 1. Now depending upon whether b 1 is 0 or not learning a 2 times b 1 helps him to find out the other share of a namely a 2 and hence the complete a.

So that is why this method of computing the secret sharing of a times b by running the two instances of OT need not maintain the privacy of the bits a and b, okay. Because through one of the OTs as a receiving party, a corrupt receiver might learn a i times b j and it may end up revealing the entire a or b depending upon what is her share or his or her share of a or b value.

(Refer Slide Time: 23:14)



So what we are going to do is we are going to retain this nice idea. The idea was nice that we expanded a times b and identified some mechanism to ensure that the two parties somehow get the cross terms without learning anything about the other party's share of the bits a and b. But we identified a potential flaw here. So what we are going to do is we are going to slightly change this idea.

And the intuition behind changing the idea is that we are now going to change the inputs of the, sender's inputs of the OTs. The receiver's inputs for the OTs, namely their choice bits will remain the same. So the second party should participate with choice bit being b 2. The first party should participate with choice bit being b 1 in the other OT instance because they are not revealed, okay.

If you see the earlier method where we identified the flaw, the choice bits they were hidden because they were coming from receiver's privacy. Now we are going to change the sender's inputs in these OT instances. So the first party where he is acting as the sender he no longer feed the input 0 and a 1 as his two messages. But rather he now feeds the messages m naught and m 1 where the message m naught is a random bit r naught.

And the second message is the masking or the one time pad encryption of his a share with r naught as the pad. And now based on the correctness property of the OT It is easy to see that depending upon whether b 2 is equal to 0 or 1, the second party or the receiver of this OT will receive this value. And now you can see that she is learning r naught XORed with a 1 times b 2.

She is no longer learning a 1 times b 2 in clear as was the case in the earlier approach. In the earlier approach a 1 times b 2 was completely learnt as a receiver of the OT receiver and that led to the problem because depending upon the value of b 2 she may end up learning about a 1 if a 1 times b 2 is learnt in clear as a receiver output, okay. But now we are not letting the second party to learn a 1 times b 2.

Instead we are letting her learn a masking of a 1 times b 2 where the mask is now completely random because the mask is the other message. And she will not learn the other message depending upon her choice bit. So depending upon her choice bit if her bit b 2 is 0, she will learn only the mask that is completely useless. Whereas if her bit b 2 is 1, she learns a masking of the cross term but the mask is not revealed.

So whatever may be the case, she will not learn anything about a 1, a 1 remains as private as possible. Now let us see the sender's input for the other OT instance. For

the other OT instance we do it symmetrically. The second party P 2, she picks a random mask r 1. The message m naught is her mask r 1 and the message m 1 is a masking of her a share with the pad r 1.

And now again depending upon the choice bit b 1 being 0 or 1, the first party P 1 either receives the mask r 1 which is useless or a masking of the cross term a 2 times b 1. But with the mask r 1 which is not known to the party P 1. And now I can rearrange the terms in the expansion of a times b into this lengthy expression.

If you see this lengthy expression then finally r naught and r naught cancels out, r 1 and r 1 cancels out and then everything reduces to only a times b. But if I rearrange the terms like this, then I can identify the shares of the P 1 and P 2 which can be now computed by P 1 and P 2. P 1 can compute this share c 1 because anyhow he has a 1 and b 1. So he can compute a 1 times b 1.

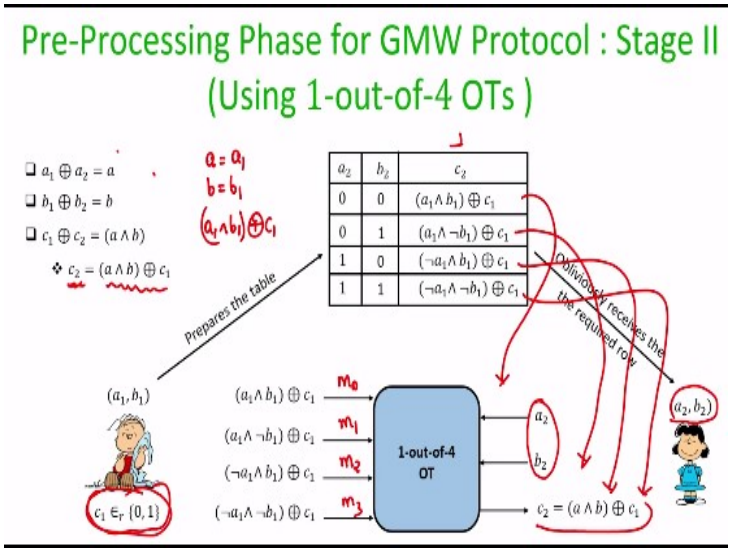
He is acting as the sender. So he has the mask r naught. And as a receiver he receive a masked version of the cross term a 2 times b 1. So he has now everything to compute the first share of c and using similar argument the second party P 2 has everything to compute the other share of c. And now you can see in this process irrespective of whether P 1 or P 2 is corrupt, nothing additional about the bits a or b is learnt.

And that maintains the privacy of your stage II. So this is now original GMW method of computing securely the cross term, okay. So remember the problem was securely computing this cross terms a 2 times b 1 and a 1 times b 2. To securely compute the cross terms we run here two instances of OT which ensures that now the each of these OT instances take care of one of the two cross terms, okay.

So the cross term a 2 times b 1 is now made available in a secret shared fashion where the share of the party P 1 is r naught and the share for the other party is a masking of this cross term with the mask r naught. Whereas the cross term a 1 times b 2 is taken care where the mask is r 1 and available with the second party.

And the first party gets a masking of this cross term with the mask r 1. And that is what is the purpose of these two instances of OT, okay.

(Refer Slide Time: 29:10)



It turns out that we can instead of running this two OTs for computing this cross terms in a secure way in a vast fashion we can run one instance of one out of four OTs okay, because and that will lead to efficiency. So let us see how we can do stage II using one out of four OTs. And advantage here will be that only one instance of one out of four routines will be executed instead of executing two instances of one out of two OTs, okay.

So the idea here will be the following. We are given this setup. This is given a and b are secret shared and our goal is to compute a secret sharing of the bit c which is the and of the bits a and b. So I can write the second share of c as a function of the and of a and b and the first share as per this expression. And based on this idea, we can let the first party P 1 to pick his share for the c value randomly, okay.

It does not know the and of a and b, but it can pick his share for the c value, the resultant c value randomly. And now it prepares a table. He can prepare a table indexed by the shares of a and b that are available with the other party. Now depending upon the shares of a and b which are held by the other party, in his mind the first party can compute the values of c 2.

Well it does not know what exactly is the value of a 2, what exactly is the value of b 2 and hence, what exactly is the value of c 2. He is just preparing this table in his mind. He knows that if the other share, if the shares of a and b that are held with the other

party are 0, right. Let us see how exactly this table is computed. So let us see how exactly the first row is computed.

If a 2 is 0 and b 2 is 0, that means the shares of a and b held by the other party are both 0, then basically this means that the value a which is secret shared is same as a 1 and the value of b which is secret shared is b 1 namely the shares of a and b held by the first party. Well first party does not know definitively this is the case or not, okay. This is a probabilistic event.

Now if the value a which is shared is a 1 and the value b which is shared is b 1 then what can we say about c 2? c 2 will be nothing but the and of a 1 and b 1 XORed with c 1 and that is what is the value of c 2 kept in the first row here. Similarly, let us see how the fourth row is prepared, okay. Similarly, you can run the argument to see the second and the third row.

If the other shares of a and b are, if the other share of a and b are 1 and 1 respectively that means a is same as a 1 compliment because 1 XORed with a 1 is nothing but a 1 compliment. And in the same way b is same as b 1 compliment. And if that is the case, then c 2 is nothing but the and of a 1 complement, XORed with c 1. And that is what is the fourth row. So once P 1 set his share of c 1 randomly he can prepare this table.

And now he participates as a sender with this four inputs. So these are his four messages; m naught, m 1, m 2, m 3. And the second party P 2 acts as a receiver for this OT instances, for this OT instance. So she has to provide two input bits because this is a instance of one out of four OT. So her selection bits will be a 2 b 2.

And depending upon the value of a 2 b 2 the correctness of the one out of four OT instance will ensure that she gets the corresponding row of this table which sender has prepared. So if say for instance a 2 b 2 is 0, then this is the value which gets transferred. If a 2 b 2 is 0 1 then this is the value which gets transferred. If a 2 b 2 is 1 0, then this is the value which gets transferred, otherwise this is the value which gets transferred.

And now you can see that indeed the XOR of c 1 and c 2 is a times b. Why the security is preserved here? If sender is corrupt, namely P 1 is corrupt, it does not learn anything about a 2 and b 2 because that are the selection bits, because they are the selection bits of your second party or the receiver. So receiver's privacy ensure nothing about a 2 b 2 is learnt.

Whereas if p 2 is corrupt, then as a receiver she only learns the message or only the rows she is interested in. She does not learn anything about the remaining three messages. And that ensures that nothing about a and b is revealed in this process.

So you can make the preprocessing phase of GMW protocol stage II more efficient using one out of four OT instances because now for each multiplication triple a b to compute the c component, only one instance of one out of four OT instance need to be executed instead of executing two instances of one out of two OTs. So with that, I end today's lecture. Thank you.