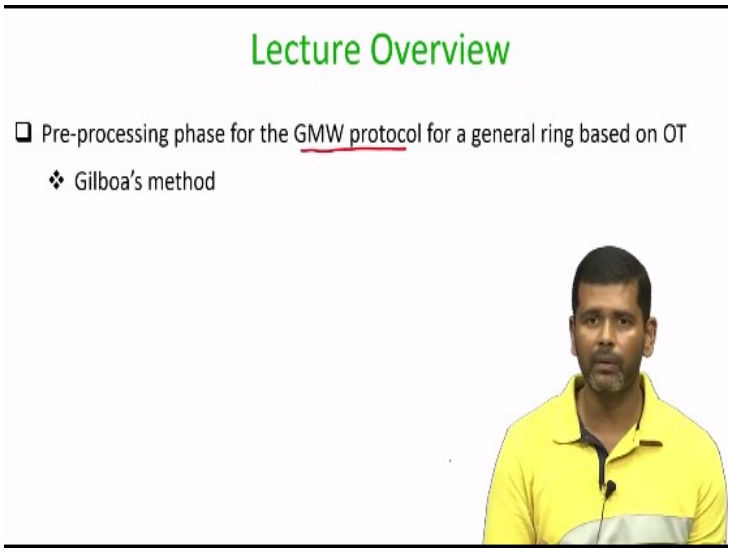
Secure Computation: Part 1
Prof. Ashish Choudhury
Department of Computer Science & Engineering
International Institute of Information Technology-Bengaluru

Lecture - 44
Pre-Processing Phase for the GMW Protocol Contd.

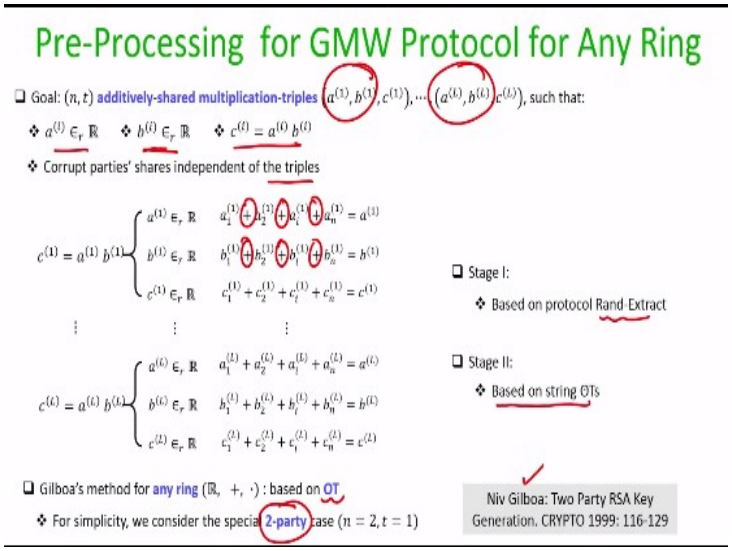
Hello everyone, welcome to this lecture.

(Refer Slide Time: 00:31)



So in this lecture we will continue discussing the GMW pre-processing phase protocol. But now we will consider it for the general ring instead of the Boolean ring and for simplicity we will stick to the 2-party case but again as we have generalized the two party solution to the n-party solution for the Boolean ring case this method also can be generalized to the n-party case.

(Refer Slide Time: 00:57)



So when we say the n-party case, the goal is the following. We now want to generate additively shared values over the rings namely additively shared multiplication triples where the A and B components should be random ring elements. The c component should be the product of a and b component and the byproduct and now mean the multiplication of the ring not the and because it need not be the Boolean ring necessarily.

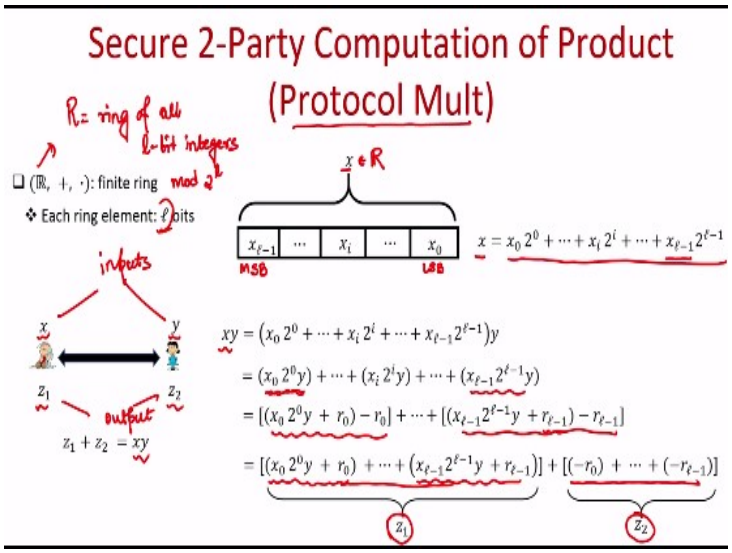
And throughout the process the corrupt parties namely the parties which are under the control of adversary which are t in number there, you should be, their shares should not reveal anything about the underlying a, b, c values, okay. And since we are now using additive secret sharing the values are shared as per the addition operation over the ring, need not be the XOR operation.

We will see the method due to Gilboa which is a very cute idea, nice idea based on OTs again and as I said earlier for the purpose of demonstration we will focus on the two party case but whatever we are going to discuss here it can be very easily generalized to the n-party case. Again the process or process of generating this shared data structure will be divided into two stages.

In stage I, we have to first generate the secret shared a and b components of all the triples. That can be done again using a randomness extraction process. And that randomness extraction process can, will easily work for the arbitrary ring as well even though we have demonstrated for Boolean ring. So I am not going into the details. We

will focus on the stage II namely, which is to get the c component of the secret shared a and b component.

(Refer Slide Time: 02:51)



So for getting into the details of the Gilboa's method, we will first see a multiplication protocol which will be useful when computing the secret shared c components. And what exactly this multiplication protocol does? Here we have two parties. And each party has a private input which is a ring element. So party 1 has the input x, party 2 has the input y, which are private ring elements available with the respective parties.

And assume for simplicity that each ring element requires l bits. So you can imagine say for instance your ring R is the ring of all l-bit integers. And all the operations are performed say modulo 2 power l. That could be one candidate ring where all the ring elements are represented by l bits. But in general, it could be any abstract ring.

But since it is a finite ring, we can make an assumption that each element of the ring can be encoded by l bits, okay. That means there are 2 to the power l elements in the ring. And the inputs are the ring elements x and y. And output that we require from this mult protocol is a secret sharing of x times y. Namely we want at finally at the end of the protocol P 1 should have a ring element z 1.

P 2 should have a ring element z 2 which are random ring elements such that the summation of the ring elements z 1 and z 2 is the product of x and y and in the process nothing additional about x and y should be revealed. Now to do this we

interpret x to be represented by l-bit. So this is the LSB and this is your MSB. So since x is a ring element, it can be represented by l bits.

So I have just focusing on the bit representation of x. And if this is the bit representation of x, then the value of x as per the expansion binary expansion will be this, LSB times 2 power 0 plus next bit times 2 power 1, ith bit 2 power i times and the contribution of the last bit the MSB bit will be 2 power l – 1, okay?

Now if this is the case, what can I say about x times y? x times y will be this expression. Now I can take y inside, because I am just using the ring axioms. I can apply the, I can distribute the plus over dot and so on. So if I take y inside then the expansion of x times y becomes this. And now what I am going to do is a volley. Let us see the first x summand here.

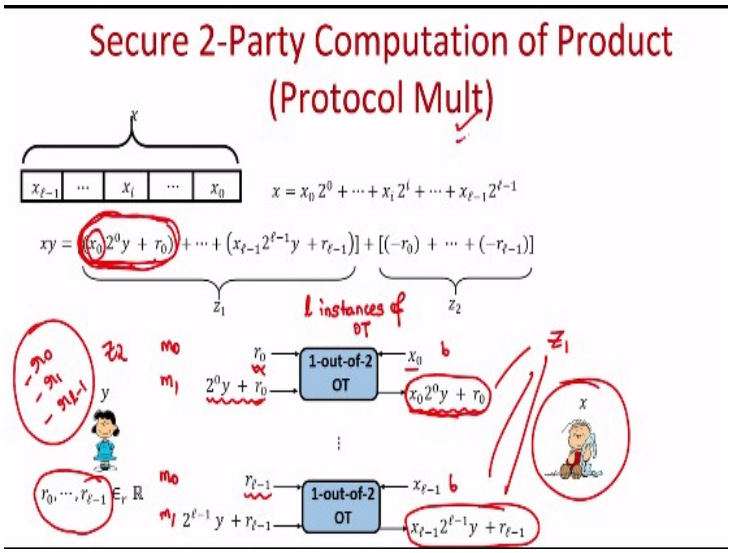
The first summand can be further expanded as the summation of these two expressions. Because here r naught is some ring element and if I add r naught to the first summand and then subtract r naught, the effect of the first summand remains as it is. And in the same way, if I consider say the last summand, if I add a random ring element and then subtract it, the overall effect remains the same.

Because the effect of this random element r sub l - 1 will not come into the picture. And now if I rearrange everything here, and take all the summands added with the random values, which I have introduced, I sum all of them together. And the other part of the expression is that I take the additive inverse of all the random values that I have added together.

And now my claim is that somehow we can set the first part of the expression here to be the share of the first party that we want, and the remaining part of the expression to be the second share for the value x times y. And indeed, if you can see here if I sum z 1 and z 2, I get x times y. So the idea behind this mult protocol will be to somehow ensure that the value z 1 is made available with party 1 and the value z 2 is made available with party 2.

But now the question is how exactly we get it done and who should select the values r naught, r 1, r l – 1.

(Refer Slide Time: 07:29)



So let us go into the details here. This is what we want to ensure, we want to generate. So party 2 here, she acts as a sender in l instances of OT. Why l instances because there are l bits of x. So she acts as a sender in l instances of OT. And the first party who has the input x, he acts as a receiver in those l instances. Well as a protocol description we can always change the role of sender and receiver, it is just for simplicity.

Since I have used to the bits of x while expanding the x times y, I am keeping the owner of the x as the receiver because the bits of x will now acts as a choice bit in this OT instances. And is all this random ring elements r naught to r of l - 1 which are introduced here, let the sender of these OT instances who is in this case is the second party, let the second party pick them, okay.

So the owner of y here pick this l random range elements and now he prepares the inputs for this OT instances. So let us see closely how these inputs are prepared here. So we want that the first summand here should somehow go in a masked fashion to the receiver, okay. So the x naught portion here is going to be the choice bit which is going to be fed by the owner of x and somehow we want this expression to go as it is, this highlighted circled thing as a expression it should go to the receiver here.

So for ensuring that to the first OT instance, the sender is feeding the messages m naught and m 1 like this. The mask is the zeroeth message and the masked version of 2 power 0 times y is the second message. And like that if I consider the lth instance of the OT, the sender's messages are these. The lth mask and the masked version of 2 power l - 1 times y. Whereas the owner of x select his choice bits to be the individual bits of the x.

And he knows the full x so he notes the bits of x so he can feed these bits as the choice bits. And now the correctness property of the OT will ensure the following. Through the first OT instance, the first party or the owner of x will receive a masked version of x 0 times 2 power 0 times y where the mask is r 0. Namely if his choice bit x 0 is 0, then he learns only the mask r naught.

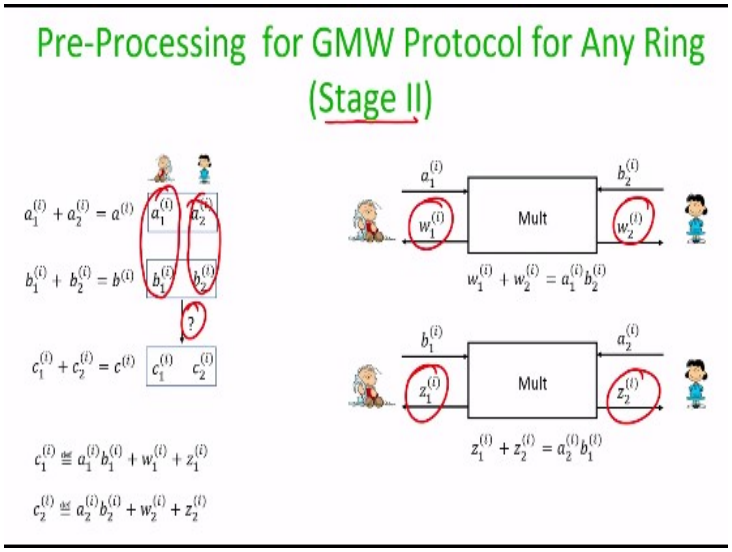
Otherwise he learns the masked version of this thing, but he would not learn both the messages. That means now we have ensured that in the z 1 which we are trying to compute or ensure that P 1 have the z 1 whatever is this highlighted thing somehow will now go as a output of the OT as a receiver to this owner of x. And in the same way, if I focus on the last OT instance as a receiver, he will obtain this value.

So now what the owner of x has to do? He sums up all the received thing in all the OT instances that will be z 1. And anyhow, all this mask values are picked by the owner of y. She can compute the additive inverse of this masks and she can sum them up and hence she will get z 2 and hence the requirement of the mult protocol are satisfied. In this process nothing about x is revealed because the bits of x are nothing but the choice bits of our receiver here.

And if the OT is secure, then the privacy of OT ensures that nothing about the choice bits are revealed and that ensures that the full x remains private. Whereas if the privacy of y is revealed, because in each instance of OT even though the input y is used, an independent pad is also getting used. And depending upon the choice bit, either the mask is received or a masked version of a power of product of y is received, but not both of them.

And for each instance of OT a random mask is used. So that ensures that if the owner of x is corrupted he does not learn anything about y. So we have now obtained a secure mult protocol.

(Refer Slide Time: 12:20)



Using it as a sub protocol we can now design the stage II of the GMW pre-processing for any ring. What will be the goal of the stage II? We have l random multiplication triples which we want to generate. Stage I has been already executed. That means the secret shared a and b components are already available. Somehow we have to ensure that the secret shared c is also made available.

And again the idea here will be the following. If I consider a 2-party case here which can be easily generalized to the n-party case, there are two cross terms which have to be somehow securely computed. Hence, the two parties run two instances of mult protocol. In the first instance, they take care of the cross terms a 1 times b 2. For that they respectively participate with inputs as the shares of a and b, okay.

And assuming that the mult protocol is secure, the mult protocol will produce outputs of the shares a 1 times b 2 for them. And in the second instance of mult they participate with the shares of b and a as their respective inputs, the mult protocol will produce random shares of the cross terms a 2 times b 1 as output.

And now the overall share for the c or the and of a and b for the first party will be whatever it can compute locally plus whatever it has received in the two instances of

mult protocol. And similarly for the other party, whatever she can compute, multiply. And whatever she has received in the respective instances of mult protocol, sum them together, that will be her share of the c value.

And it is easy to see that the security of a and b is preserved in this case as well. And this method can be easily generalized now to the n-party case because now if you have n parties there will be order of n square number of cross terms where for each pair of parties there will be two cross terms so they have to run two instances of the so called mult protocol that we have designed.

And now the overall share for c for each party will be a i times b i plus whatever they receive as a receiver in the respective instances of mult, okay. So that method can be easily generalized. I am not going to the details. I am leaving the details as an exercise for you. Thank you.