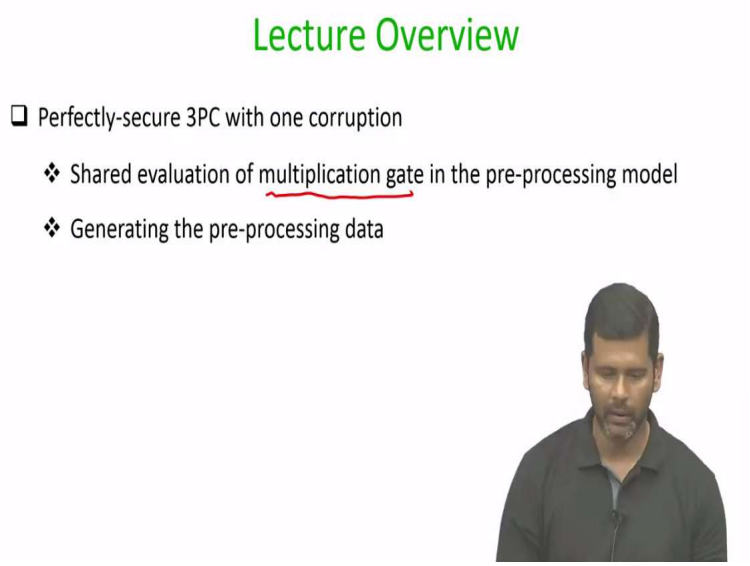
Secure Computation: Part 1
Prof. Ashish Choudhury
Department of Computer Science
Indian Institute of Science – Bengaluru

Lecture – 32
Perfectly Secure 3PC Continued

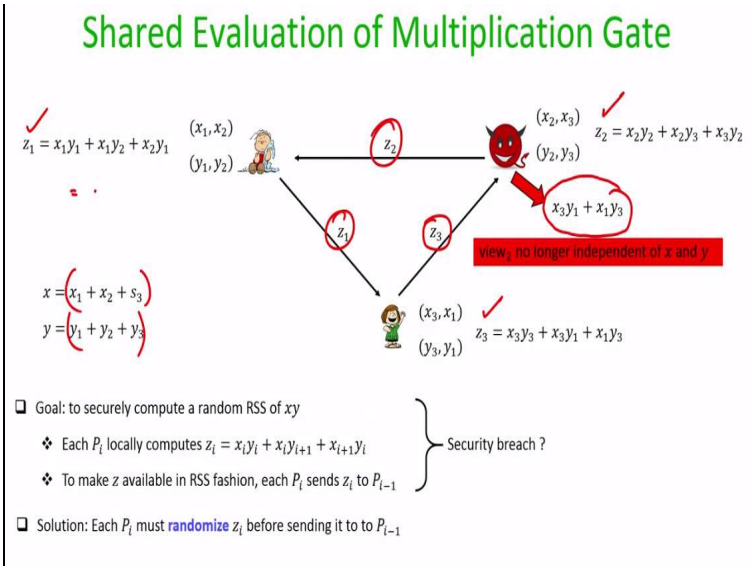
Hello everyone. Welcome to this lecture.

(Refer Slide Time: 00:31)



So, in this lecture we will continue our discussion on the efficient perfectly secure 3PC computation 3 party computation with one corruption and the focus of this lecture will be how to do the shared evaluation of multiplication gate in the pre processing model and then we see that how to generate that pre processing data.

(Refer Slide Time: 00:57)



So, just to quickly recap in the last lecture we saw that if we are in a scenario where the inputs of a multiplication gate RSS format and we want the multiplication gate output also to be available in RSS secret shared fashion and in the process do not want to reveal any information about x and y and along with that we also want to minimize the communication. So, the protocol that we proposed in the last lecture was the following.

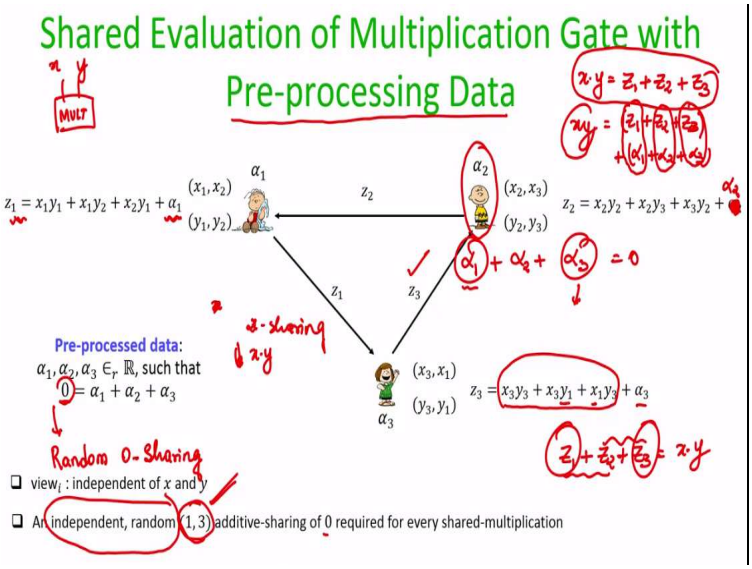
We ask each party to basically compute summation of three of the summands in this expansion of x and y. So, P_1 can find out the summation of three of the summands call it Z_1 P_2 computes summation of three of the summands call it Z_2 and P_3 computes the summation of three of the summands call it Z_3 and we thought that okay now we know the summation of Z_1, Z_2, Z_3 is x times y why not let the parties just communicate the missing piece to other party to ensure that x times y is secret shared in a replicated secret shared fashion.

But in the last lecture we saw that this leaks information about the inputs of the multiplication gate. So, for instance if P_2 is corrupt then we can no longer say that its view is independent of x and y because now it is learning an information about y_1 and x_1 and the rectification for this problem is that somehow we have to ensure that each P_i should randomize z_i before sending it to the neighbor.

So, P_1 should randomize z_1 and change it to some other value before sending it to P_3 so that if P_3 is corrupt it does not learn anything additional about x or y. In parallely P_3 when sending Z_3

should change it, should add some randomness and make it some other value such that a corrupt P_2 does not learn anything and similarly when P_2 is sending to P_1 it should randomize Z_2 so that a potentially corrupt P_1 does not learn anything about xy and such that in this whole process we do not require any other communication.

(Refer Slide Time: 03:29)



Well, let us see how that magic can happen and for that we will assume that the parties have some pre processing data. So, in the last lecture we already saw that if you want to secret share some value the dealer need some pre processing data to be generated in a pre processing phase and now for the multiplication gate also we will see that if some pre processing data is available.

Then the parties can very simply evaluate the multiplication gate in the RSS shared format and it turns out that the pre processing data that we require for the multiplication gate is precisely similar what we require for making the actual sharing protocol faster. Namely we require random zero sharing that means if there is a multiplication gate which needs to be secret shared whose inputs are x and y.

We will assume that in the pre processing phase a random zero sharing has been already generated, how it will be done we will see it later, but we will assume that one such random zero sharing has been already generated that means if in your actual circuit you have million of multiplication gates

then in the pre processing gate the random zero sharing should have been generated million number of times.

So, we will see how exactly that pre processing happens, but for the moment assume that such a pre processing data is available. Now what the parties can do is the following. We already have identified three terms here Z_1, Z_2, Z_3 we know that x times y is the summation of Z_1, Z_2 and Z_3 we already know this. Now what can I say about the summation of Z_1, Z_2, Z_3 and α_1, α_2, α_3 well that will be same as x dot y.

Because α_1, α_2, α_3 has the property that as a sum it produces the value 0. Now in this new summation the term $Z_1 + \alpha_1$ can be computed by P_1 $Z_2 + \alpha_2$ can be computed by P_2 and $Z_3 + \alpha_3$ can be computed by P_3 and now $Z_1 + \alpha_1$ is now a random piece of information for anyone who gets it because α_1 would not be known to that party.

And that precisely what I meant by randomization. So, what P_1 will do is P_1 will now add α_1 to the earlier Z_1 and let me still call Z_1. Similarly, P_3 will add α_3 to her earlier α Z_3 and call the resultant value as Z_3 only and similarly the Z_2 is now changed to this value and now whatever we proposed earlier let P_1 communicate Z_1 to P_3, P_3 communicate Z_3 to P_2 and P_2 communicate Z_2 to P_1 let it happen.

And now it is easy to see that the new Z_1, new Z_2 and the new Z_3 they sum up to x times y only and indeed P_1 has these two pieces, P_2 has these two pieces and P_3 has the first and the third piece. So, x dot y is available in a replicated secret shared fashion. Now, let us try to understanding that whether the privacy of x dot y x and y is preserved now in this new process.

Again let us take the same scenario say P_2 is corrupt earlier it was receiving Z_2 without this α_2 and that is why this problem was occurring. It was receiving Z_3 without this α_3 and that is why it was learning something about y_1 and x_1, but now in the new thing if α_3 is also summed up and given to him. So, it is like P_2 is seeing an OTP encryption of this value where the pad is α_3.

And α_3 is not known to P 2 because even though P 2 knows what is α_2 it does not know what is α_1, it does not know what is α_3. These are unknowns for him. Even though it knows that they sum up to 0. So, it could be any α_3 which has been used by P 3 such that corresponding to that α_3 there would have been an α_1 which along with the α_2 which is held by P 2 would have produced the value 0 that means view 2 is now completely independent of x and y and hence it can be easily simulated by any simulator.

So, that ensures the privacy of x and y in this new process and now you see that this method is so efficient when you want to evaluate the multiplication gate each party just needs to send one ring element and that too only to one party. This is unlike your GRR degree reduction or Beaver's circuit randomization method where for evaluating each multiplication gates two values needs to be publically reconstructed namely the masking of the gate inputs of the multiplication gates have to be publically reconstructed.

And then you apply the Beaver's linear function to get the multiplication gate output in the secret shared fashion. Here, we do not need to publically reconstruct any value just do a local randomization of your summation of summands and sent it to one of your neighbors that is all. It is such a simple protocol and that is why it is very, very efficient method. Now as it is the case for any pre processing based protocol if you want any security not just any perfect security for the purpose of security for every multiplication gate an independent and randomly chosen additive sharing of 0 has to be used from the pre processing phase.

So, if you have L number of multiplication gate you need to have L number of independent random 1, 3 additive secret sharing of 0, y 1, 3 secret sharing of 0 because even though the value 0 is known to be shared by the parties or the corrupt parties the bad party will have either α_1 or α_2 or α_3 namely one of the α values. It would not be knowing the remaining two α values.

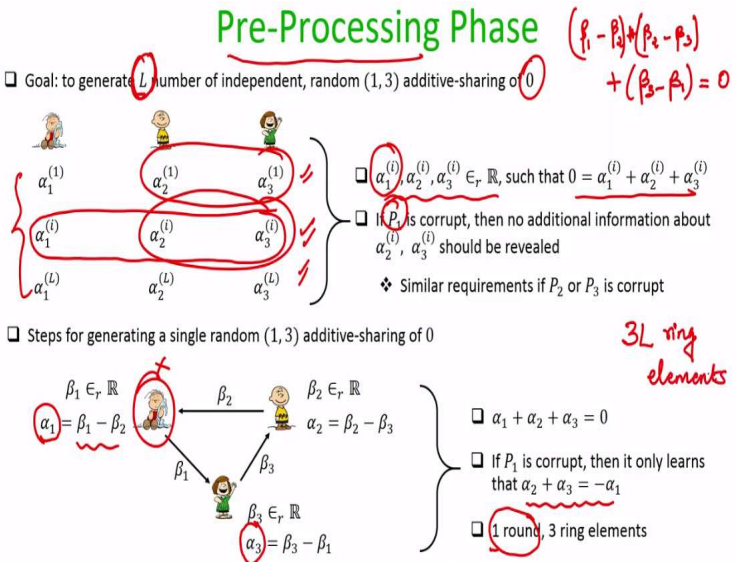
So, to reconstruct the full vector of α values we need the collaboration of 2 parties so that is why 1, 3 additive sharing of 0. So, that is a way we can perform the shared evaluation of multiplication gates assuming we have a pre processing data. So, now the question is how exactly we generate the pre processing data. By the way before proceeding further the process of this shared evaluation

of multiplication gate where each party adds an αcomponents to the summation of its summands it can be compared with the GRR degree reduction method where the parties once they obtain 2t sharing of x dot y or non random 2t sharing of x dot y converts it into a random 2t sharing of x dot y.

And how they convert it into a random 2t sharing of x dot y? We ask each party to just add a share of 0 where the vector of zero shares lie shares lie on a random 2t degree polynomial. So, we have done the same thing here x dot y is the summation of old Z_1, Z_2, Z_3 now we just add α_1, α_2, α_3 here component wise the shared value remains the same because α_1, α_2, α_3 adds up to 0.

So, you can compare the idea that we have used here with that degree reduction method used in the GRR degree reduction.

(Refer Slide Time: 11:58)



So, now we have to focus on the pre processing phase. Our goal is to generate many number of additive sharing of 0 independent of each other. So, imagine it is basically we need to generate shared values like this. So, we have L shared values here. So, let us focus on the ith vector here. So, what is the property of the ith vector? The property of the ith vector here is that they are random elements from the ring such that they sum up to 0.

And we have to generate this data structure this system of values in such a way that if one of the parties is corrupt then it only learns about its only own αvalue or any share of the 0. It does not learn anything additional about the shares of the 0 that held by the remaining 2 parties. Of course, it will know that along with his own share the summation of the shares of the remaining 2 parties will sum up to 0 that much information is allowed to (()) (13:12).

Other than that during the generation of this system of values no additional information should be revealed. So, for instance, if P_1 is corrupt it will have access its view will consist of all the first components here in all the vectors, but from its viewpoint remaining two components of each two vectors could be any two values which along with his own components sum up to 0.

So, that is a privacy requirement while generating this system of values. So, now let us see how we can generate one such vector of values the same process can be done, can be executed in parallel L number of times if you want to generate L number of such random sharing of 0. So, again each party picks three random values locally let me call them as β values.

So, P_1 picks β_1, P_2 picks β_2 and P_3 picks β_3. You can imagine that they are doing this process L times because finally their goal is to generate L number of values randomly. Now, what I can say about this summation $\beta_1 - \beta_2 + \beta_2 - \beta_3 + \beta_3 - \beta_1$ all of them when summed up gives you the value 0 and we basically want to generate a secret sharing of 0 and that is what is the protocol based on this idea.

The protocol is based on this idea so now what P_2 does is whatever value it has generated it sends to P_1 whatever value P_1 has generated it sends to P_3 and whatever value P_3 has generated it sends to P_2. So, again the same circular order of communication is followed. So, remember in all the protocols that we have seen for this 3PC computation based on replicated secret sharing we maintained this very nice invariant that wherever the communication is required a party has to send message only to one of its neighbor.

It could be either the right neighbor or the left neighbor. So, depending upon in what order means it could be based on clockwise ordering or anti-clockwise ordering. I am following the anti-

clockwise ordering, but it is up to you. The parties can decide what ordering to follow and that they will communicate according to that ordering. So, now once the parties exchange these values we can set the α values to be the following.

P_1 can set his α component to whatever β value it has picked and whatever β value it has received difference of those two values. P_3 will set his α component to be whatever β component he has randomly picked and whatever β component he has received the difference of those two values and same for P_2 and now it is easy to see that α_1, α_2, α_3 sums up to 0 and again let us see whether the privacy properties that we required is satisfied or not.

So, imagine if this P_1 is corrupt and whatever I am discussing here holds for the case when any of the remaining 2 parties gets corrupt that means if P_2 gets corrupt and then also the same argument holds and so on. So, if P_1 gets corrupt then of course it will know $\alpha 1$ and of course it will know since it knows β_1 and it is receiving β_2 it knows two of the β values, but it would not be knowing the third β value because that is picked by the third party and third party is not under the adversary's control.

So, that means if adversary is corrupt adversary in this case is P_1 then the only thing which it learns is the following that the summation of the remaining two α components is actually minus of his own α component, but that much information is anyhow allowed to be revealed from each of the vectors of α values which we want to generate. Apart from that no other additional information is revealed in this process of generating α_1, α_2 and α_3.

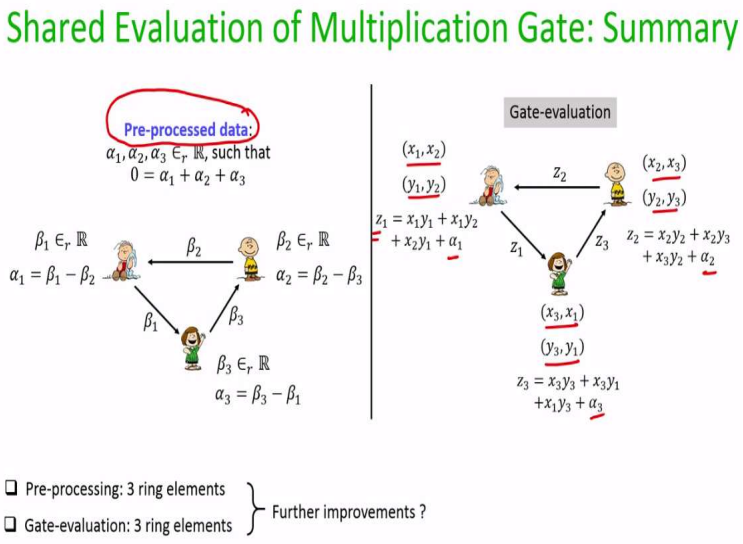
And that is why the view of the adversary will be independent of the inputs of the honest parties and that shows that a pre processing here is so simple we do not have to do any degree reduction nothing of that sort and this is unlike your Beaver's circuit randomization method where to generate the multiplication triples we have to run more complex protocols based on randomness extraction, degree reduction polynomials extrapolation, interpolation and so on.

Here we do not need to do anything parties just pick up random values and just send those random values to one of those neighbors, respective neighbors and that is all. So, that means if you want

to generate the pre processing data L number of times it will require only one round because I have generated here only one random additive sharing of 0, but to generate L number of random additive sharing of 0 the same process can be executed L number of times.

That means P 1 has to pick L number of β values and send to P $_3$ in parallel P $_3$ has to pick L number of β values randomly and send it to P $_2$ and so on. So, it will be only one round of communication and the total number of communication will be (()) (19:13) ring elements.

(Refer Slide Time: 19:21)



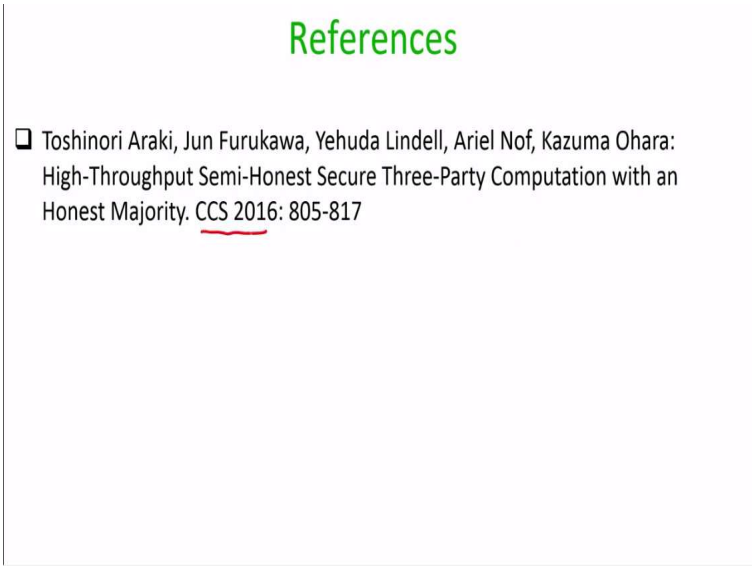
So, now you have all the building blocks to get the full MPC protocol for the 3 party setting. I am not going to into the full details of the 3 party protocol because now you know how to secret share the inputs, you know how to reconstruct secret shared output, you know how to do the pre processing, you know how to do the evaluation of the linear gates and you know how to do the evaluation of the multiplication gates.

By stitching all these components together we now know how to perform the shared circuit evaluation. So, I am not giving the full protocol for the shared circuit evaluation its privacy proof all those things are now easy to come up with because we have seen component wise the privacy of each protocol is preserved. So, what I want do here is I want to summarize the multiplication process here.

So, this is the way we do the multiplication so if there is one multiplication gate then the pre processing phase a setup has to be generated namely a random sharing of 0 with threshold one and assuming that setup is there than in the gate evaluation when the actual inputs of the multiplication gates are available in a replicated secret shared fashion then the parties can just locally first compute the summands that they are supposed to compute and randomize it and send it to the designated neighbor.

So, that means in the pre processing phase we require 3 ring elements to be communicated and actually at the time of gate evaluation again 3 ring elements have to be communicated. Now the question is can we further improve this communication and that will be the focus of our next lecture.

(Refer Slide Time: 21:14)



So, today's lecture is again based on the same paper which we have followed in the last lecture. Thank you.