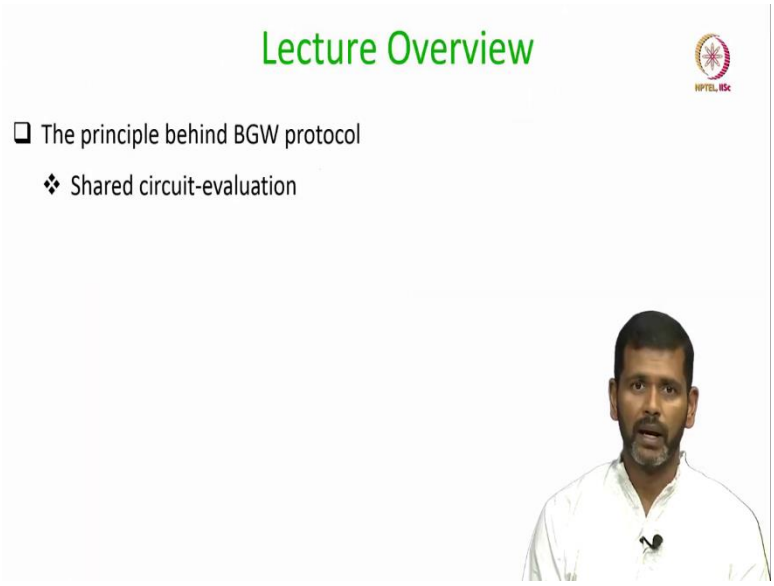
Secure Computation: Part II
Prof. Ashish Choudhury
Department of Computer Science and Engineering
Indian Institute of Information Technology, Bangalore

Lecture - 31
The BGW MPC Protocol for Passive Corruptions: Recap

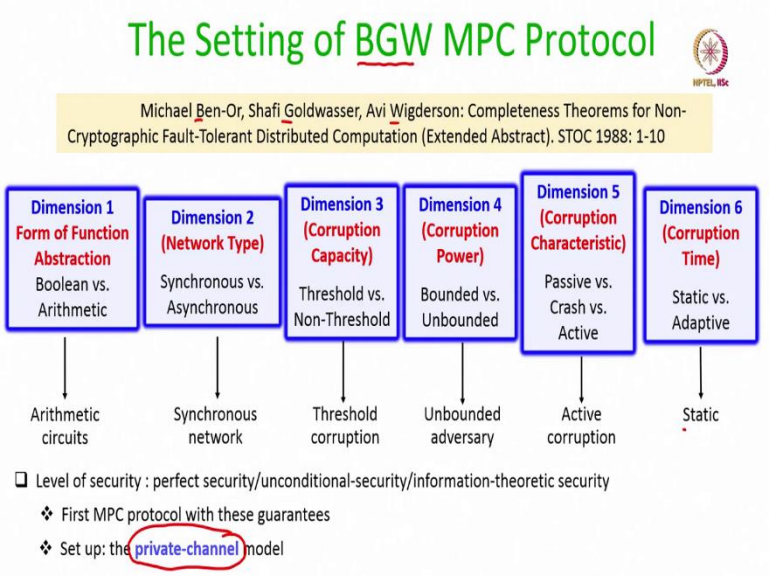
Hello everyone, welcome to this lecture. So, in this lecture, we will have a Recap of the BGW MPC Protocol for Passive Corruptions. So, for the full details of the BGW MPC protocol for passive corruptions, you are referred to the secure computation part I course; we will see the exact reference at the end of the lecture.

(Refer Slide Time: 00:44)



So, the plan for the lecture is as follows; we will try to understand the principle behind the BGW MPC protocol, namely the shared circuit evaluation.

(Refer Slide Time: 00:55)



So, what is the setting of the BGW MPC protocol? So, BGW MPC protocol was invented by Ben or, Shafi Goldwasser and Avi Wigderson, so that is why the name BGW; it was published in 1988. And what is the setting considered in this MPC protocol? So, recall that the problem of secure computation or MPC can be studied in various domain, various dimensions depending upon how you are abstracting your underlying function to be securely computed.

What is the type of underlying network, what is the corruption capacity of the adversary, whether the adversary is computationally bounded or unbounded, whether the adversary is corrupting in a passive fashion or malicious fashion and whether the adversary static or adaptive?

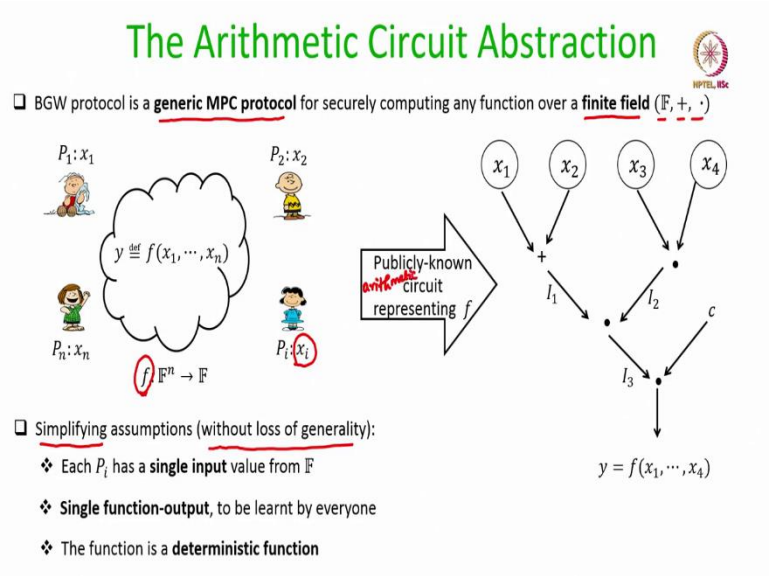
So, the exact dimensions studied captured in the BGW protocol is as follows. So, it is for arithmetic circuits; for the synchronous communication model, where the corruption capacity of the adversary is threshold and adversary is computationally unbounded and the protocol, the BGW protocol, was designed both for passive as well as malicious or active corruptions.

And it considered static adversarial setting; but the security of the BGW protocol can also be proved against the more powerful adaptive adversary. So, the level of security which is provided or achieved by the BGW protocol is perfect security, often called as

unconditional security or information theoretic security and it is the first MPC protocol with these guarantees.

So, the setup assumed in the protocol is that of private channel model, namely we assume that every pair of parties have a secure channel between them over which they can securely communicate. We can instantiate this private channel model by running a PSMT protocol if your underlying communication network is an incomplete graph.

(Refer Slide Time: 03:26)



So, let us try to understand the arithmetic circuit abstraction first. So, as I said earlier BGW protocol is a generic MPC protocol, where the underlying function could be any function and we assume that the function which is to be securely computed is a function over some finite field, with some abstract $+$ and abstract $\cdot$ operations.

And we make several simplifying assumptions while presenting the BGW protocol; I stress that all these are simplifying assumptions and they hold without loss of generality. So, what are those assumptions? First, every party has a single input for the function.

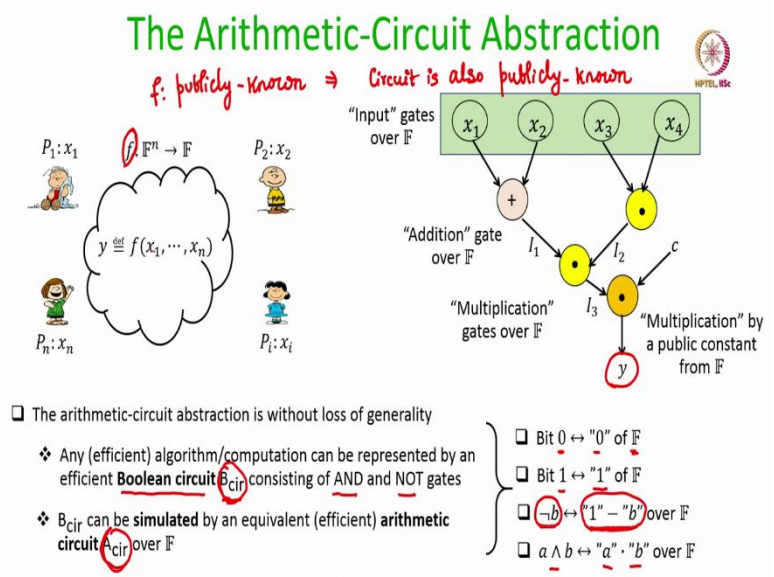
So, we assume that the function f which the parties want to securely compute is an n-ary function, namely it takes n inputs; one input is going to be provided by each party. So, the input of the ith party will be some field element x_i, which will be a private input known only to the party P_i.

And the function is a single output function, but the function output is supposed to be learned by everyone. So, these are two simplifying assumptions, of course the protocol can be easily generalized for the case when every party has more than a single field element as input and where the function can have different outputs for different parties.

And the third simplifying assumption is that the function is a deterministic function; that means internally during the computation, internally the function f computes its output as a deterministic function of its input, that means the output will remain the same if the inputs are same. So, as I said these are some simplifying assumptions without loss of generality.

So, even if every party has more than one input or if there are several outputs of the function, say n outputs where the ith output is supposed to be known only by the ith party or if the function is a randomized function, all those things can be easily handled by the BGW protocol. So, since the function f is an arbitrary function, we assume that the function f is represented by some publicly known arithmetic circuit over the field.

(Refer Slide Time: 06:28)



And that circuit will have some gates over the field. So, let us try to understand the various gates here. So, the first layer of gates in the circuit will be the input gates, namely the respective input values of the parties.

Then in the circuit, we can have some addition gates over the field. So, for instance this is an addition gate here, where the output of the gate is the addition of the inputs, where the addition is the underlying + underlying addition operation of the field. The circuit can also have some multiplication gates over the field; it can also have some gates where a multiplication is performed by some publicly known constant from the field and then we have the output value.

So, depending upon what exactly is the function f which the parties want to securely compute; we have a corresponding arithmetic circuit representation for the function f. And that arithmetic circuit will be publicly known, because the description of the function f is publicly known. So, the function f is publicly known that, automatically implies that the circuit is also publicly known.

Now, you might be wondering is it possible to always represent any computable function f as an arithmetic circuit over some finite field, and the answer is yes; this arithmetic circuit abstraction is without loss of generality. Why so? So, when I say the parties want to securely compute a function, it is as good as saying that the parties want to perform some computation or run some algorithm that is abstracted by a function.

Any algorithm or computation can be represented by an Boolean circuit consisting of some universal gates say the NAND gates and the NAND gate can be composed of AND negation gates or NOT gates. So, that Boolean circuit can be easily simulated by an equivalent arithmetic circuit over the field; that means whatever is the computation which is supposed to be carried out by that underlying Boolean circuit; corresponding to that Boolean circuit we can always find an arithmetic circuit, which also performs the same computation.

How can we do that? Well, we can say the following: we take a field and that field will have the additive identity element 0. So, the bit 0 in the Boolean circuit can be mapped to the additive identity 0 of the field; in the same way the field will have a multiplicative identity element 1. So, the bit 1 in the Boolean circuit can be mapped to the identity element 1 of the field.

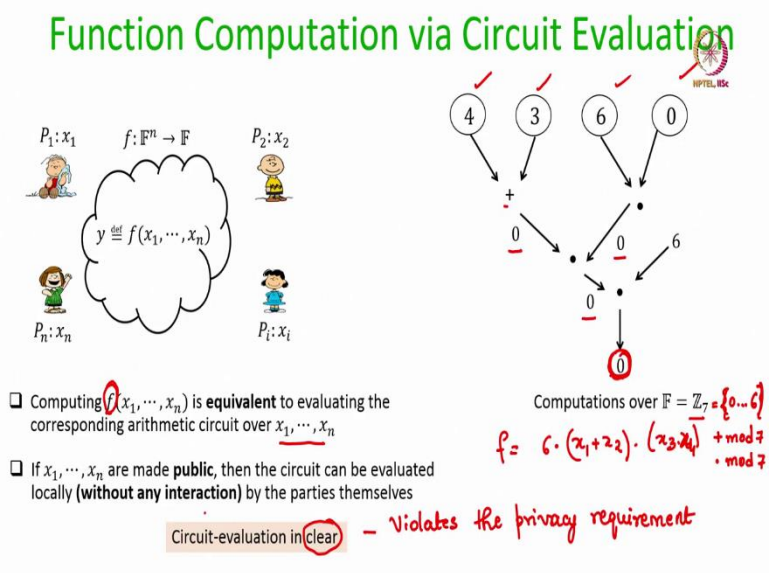
And now wherever in the Boolean circuit, we have a negation gate that can be simulated by performing this operation over the field; that means corresponding to every NOT gate, we can write down a gate over a field, namely a subtraction gate. And remember over the

field, there is no subtraction; subtraction is nothing, but adding with the additive inverse. So, we can simulate the effect of $\neg b$ as $1 - b$ using an arithmetic gate over the field with a $+$ gate performing this operation.

And wherever there is a Boolean and gate involving the bits a and b; in the Boolean circuit that can be simulated by an arithmetic gate, where we perform the multiplication of the mapped a and mapped b. So, that means corresponding to every Boolean circuit B_{cir}, I can write down an arithmetic circuit where the size of the Boolean circuit and the arithmetic circuit will be almost the same, there would not be too much of blow up.

That means if the original Boolean circuit is an efficient side circuit; that means it represents an efficient computation, then the corresponding arithmetic circuit is also an efficient circuit representing an efficient or polynomial time computation over the field. So, that is why this arithmetic circuit abstraction is without loss of generality; there is we are I am not making any sophisticated assumption by making by assuming that the circuit is represented by some arithmetic circuit over a finite field.

(Refer Slide Time: 12:08)



It turns out that computing the function f over the inputs of the parties is equivalent to evaluating this arithmetic circuit over the inputs $x_1 \ldots x_n$; why so? This is because if the parties make their inputs $x_1 \ldots x_n$ public; then anyone can compute the output of the function by evaluating each gate.

Namely if the values of x_1, x_2, x_3, x_4 is known publicly and then since the circuit is publicly known, everyone can find out what will be I_1, everyone will be able to find out what is I_2, everyone will be able to find out what is I_3 and then everyone will be able to find out what is y which is the corresponding function output.

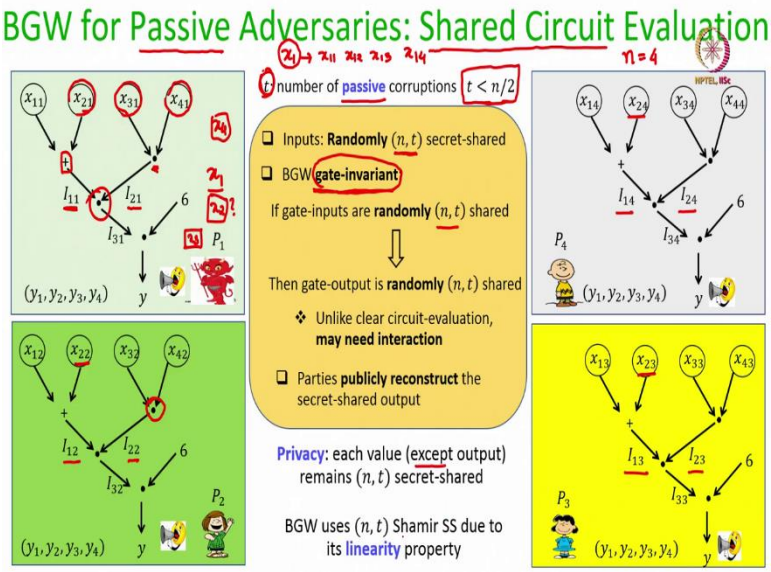
So, this is called circuit evaluation in clear. Why circuit evaluation in clear? Because the entire computation is performed over clear values; clear values in the sense that all the values during the circuit evaluation right from the input to the intermediate values are known to everyone. So, for instance if I take this example circuit which represents the computation, $x_1 + x_2$ and then you have product of x_3 and x_4 and then that is multiplied by 6, this is the function f.

Now, if the parties publicly declare their inputs to be 4, 3, 6 and 0, assuming that the all the inputs are over the field $\mathbb{Z}_7$; $\mathbb{Z}_7$ means it has the elements 0 to 6 and all the addition operations are addition modulo 7 and all the multiplication operations are multiplication modulus 7, then anyone can find out what is the value of y. So, the value of I_1 will be 0, because $4 + 3$ over the fields $\mathbb{Z}_7$ will be 0; 6 into 0 will be 0, 0 into 0 is 0 and 0 into 6 is 0. So, 0 is the function output.

So, this is called circuit evaluation in clear. So, what we understood here is that, evaluating sorry computing the value of the function is equivalent to evaluating the arithmetic circuit over the inputs of the parties. But this process of circuit evaluation in clear violates the privacy requirement.

Because the inputs of all the parties are publicly known, the inputs of all the parties are publicly known, but MPC requires that the part inputs of the parties should be remain should remain as private as possible. So, this is not the way to perform secure multi party computation.

(Refer Slide Time: 15:36)



So, the BGW protocol performs the circuit evaluation in a different way and the form of circuit evaluation done in the BGW protocol is called as the shared circuit evaluation. So, here is the idea of the shared circuit evaluation assuming that adversary is a passive adversary. And what is a passive adversary? By passive adversary I mean that the corrupt parties who are under the adversary's control, they do not deviate from their protocol instructions.

Whatever instructions they are supposed to follow as per the protocol, they follow that; they do not deviate from that. So, it is a weaker form of the adversary compared to the byzantine or malicious adversaries. Of course, we want to design the BGW protocol for malicious adversaries, which we will design soon; but right now, I am trying to explain you the idea used in the BGW protocol assuming that we are in the passive corruption model.

So, this is so, I am demonstrating the idea assuming we have four parties here, out of which t could be corrupt in a passive fashion and where $t < \frac{n}{2}$. You might be wondering why this number $t < \frac{n}{2}$; again in the first part of the course we have proved that, this condition is a necessary condition if you want to design a perfectly secure MPC protocol in the presence of t passive adversaries.

So, that is why if you have $n = 4$ parties, at most one of those four parties can be passively corrupt; then only the BGW protocol will provide you all the guarantees. If more than one party gets passively corrupt and the protocol may not provide you the required security guarantees. So, the parties will not be knowing well in advance who will be the t corrupt parties; they only will have a bound on the number of corrupt parties, but they will not be knowing the exact identity of the t corrupt parties.

So, for instance it could be the first party who get corrupt or who get corrupted during the protocol execution, or it could be the second party, or it could be the third party or it could be the fourth party. So, instead of performing the circuit evaluation in clear, the BGW protocol will perform shared circuit evaluation and where each gate will be evaluated in a shared fashion. What that, what does that mean?

So, to begin with the inputs of respective parties will be secret shared by the corresponding input owner. So, what does that mean? So, P_1 owns the input x_1; it will act as a dealer and invoke an instance of t out of n secret sharing scheme and secret share it is input. So, it will give a share of x_1 to second party, to the third party, to the fourth party and one share it keeps with itself. So, let us call the shares of x_1 as x_{11}, x_{12}, x_{13} and x_{14}; these shares are computed by P_1, because P_1 is the owner of x_1.

So, it is now not giving the value of x_1 to any single party; but rather it is giving a share of x_1 to every party and those shares are computed randomly by running an instance of t out of n secret sharing scheme. In the same way the second party who owns the input x_2, it computes four shares for its input x_2 and one share it provides to each party.

So, the first party will be provided a share x_{21}, the second party will be provided with a share x_{22}, the third party will be provided with a share x_{23} and the fourth party will be provided a share x_{24}. Same is done for the input x_3 and the same is done for the input x_4. Now, let us stop here for a moment and try to understand how much information an adversary who can corrupt up to t parties, learn about the inputs of the other honest parties.

So, it turns out that adversary will learn at most t shares corresponding to the inputs of the honest parties; but those shares correspond to an instance of t out of n secret sharing scheme and privacy property of t out of n secret sharing scheme guarantees that even if

up to t shares are given, leaked, compromised, adversary will not learn what exactly was the underlying secret.

That means for instance if P_1 gets corrupt; then of course P_1 will know x_1, because x one is the input of P_1. So, it will know its own input, but it will not be knowing x_2; because for x_2 it will only learn the share x_{21} and the probability distribution of x_{21} will be independent of the exact value of x_2.

So, adversary will not be able to tell what is x_2, adversary similarly will not be able to tell what is x_3; because corresponding to x_3, adversary learns only a single share here and in the same way adversary will not be able to tell what is x_4.

Because corresponding to x_4, the adversary learns only a single share. So, that means if the inputs are secret shared in this way by the respective input owners, then the inputs of the honest parties; by honest parties I mean the parties who are not under adversary's control, their inputs remain private, they are not leaked.

Now, once the inputs for the function are secret shared, the parties next maintain the following BGW gate invariant for each gate in the circuit. They try to evaluate the gate in a secret shared fashion, where the gate invariant is the following. If the inputs of the gate are secret if the inputs of the gate are secret shared in a t out of n secret shared fashion; then the parties, then the protocol ensure that the parties have t out of n secret sharing of the gate output.

That means the inputs of the gate were not available in clear and so, is the output of the gate. For the inputs of the gate each party had a single share and even for the output of the gate, somehow each party will have a single share, such that the vector of shares for the inputs of the gate and for the output of the gate correspond to an instance of a t out of n secret sharing. So, what does that mean?

So, the first gate in this circuit is this $+$ gate; if x_1 and x_2 would have been provided in clear, as it would have happened during circuit evaluation in clear, everyone would have learned what is the value of I_1, but now no single party knows the full value of x_1 and x_2, but everyone has a share for x_1 and a share for x_2. So, using the BGW protocol, the gate invariant will be maintained. How exactly that gate invariant is maintained? That is the crux of the BGW protocol and that we will discuss later.

But now what happen what is going to happen is, each party will locally each party will perform some operations on its shares of x_1 and x_2 and that will ensure that each party gets a share of the intermediate value. So, that means the collectively the vector of shares I_{11}, I_{12}, I_{13} and I_{14} correspond to a t out of n secret sharing of the exact intermediate value I_1.

In the same way the gate invariant will be maintained for the next gate namely for this multiplication gate, the parties will perform some operation on their respective shares for the inputs of this multiplication gate. And somehow the parties will get respectively their shares for the output of this multiplication gate; namely P_1 will have I_{21}, P_2 will now have I_{22}, P_3 will have I_{23} and P_4 will have I_{24}, such that this vector I_{21}, I_{22}, I_{23} and I_{24} corresponds to t out of n secret sharing of the exact intermediate value I_2.

And then they go to the next gate and again following the BGW protocol; they maintain the invariant for the next gate and like that once all the gates are evaluated as per the BGW protocol in this secret shared fashion, each party will be now available with a share for the output of the function.

So, P_1 will have a share y_1, P_2 will have a share on share y_2, P_3 will have a share y_3 and P_4 will have a share y_4. And once the shares for the output value are available; the parties exchange the shares, their respective shares for the output value by making those shares public. And once all the shares of the output value are made public, the parties can apply the reconstruction algorithm, or the reconstruction function of the underlying secret sharing and learn the function output.

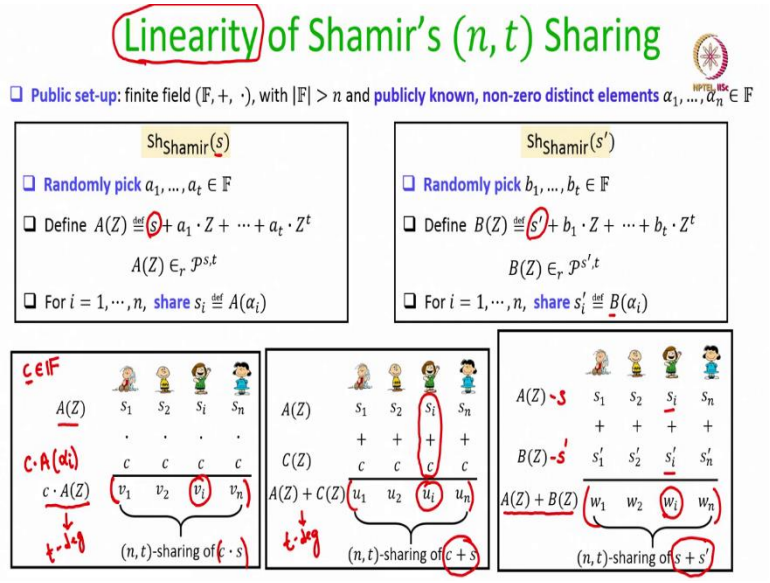
So, now you might be wondering how exactly this gate invariant is maintained; well as I said earlier that is the crux of the BGW protocol, maintaining that invariant may require interaction among the parties depending upon what is the type of t out of n secret sharing is used and what is the type of the gate, those details will be coming later. Now, intuitively this protocol maintains the privacy property, even if t out of the n parties get corrupt; because throughout the interaction, each value except the output value y remains secret shared in a t out of n secret shared fashion.

That means for each value namely $x_1, x_2, x_3, x_4, I_1, I_2, I_3$ right all those values each party will have only one share and since adversary could corrupt up to t parties; adversary will

have t shares for each of those values and those t shares will not reveal the exact value of those values, the exact value of those $x_1, x_2, x_3, x_4, I_1, I_2, I_3$.

So, that ensures that the privacy is maintained; this is not like your circuit evaluation in clear. And interestingly the BGW protocol uses the t out of n Shamir secret sharing scheme, which we had seen earlier due to its linearity property. So, we will see what the linearity property is; due to this linearity property while maintaining the gate invariant, the parties need not have to interact while evaluating the linear gates in the circuit.

(Refer Slide Time: 27:50)



So, what is the linearity property? So, recall the Shamir's secret sharing scheme is as follows; if you have a secret s over the field which needs to be secret shared and if you have a publicly known setup, namely you have n field elements which are publicly known, which are distinct and nonzero, then to share a secret s what the dealer can do is the following. It can pick a random t degree polynomial whose constant term is the secret to be secret shared and the share for the ith party is the evaluation of the polynomial or the value of the polynomial at α_i.

Now, imagine there is another dealer who has secret shared a value s' using an instance of Shamir secret sharing; for that it has picked a random t degree polynomial, let me call that polynomial as the B polynomial. The constant term of the B polynomial is s' and all other coefficients are randomly chosen and the shares for the secret s' is generated by evaluating the B polynomial at $\alpha_1, \alpha_2, \ldots, \alpha_n$

So, imagine there is a publicly known value c from the field and every party multiplies that value c with its share of s; that will give a vector of values vector of n values, where the ith component of that vector will be available only with the ith party. Now, collectively if I view this vector of new values $v_1, v_2, \dots, v_i, \dots, v_n$; what does it constitute? It constitutes a t out of n secret sharing for the secret $c \cdot s$

This is because if $s_1, s_2, \dots, s_i, \dots, s_n$ constitutes a vector of Shamir shares for the secret s, namely those values were lying on the polynomial A; then I can say that the values $v_1, v_2, \dots, v_i, \dots, v_n$ lies on a t degree polynomial c times the A polynomial. So, the c times A polynomial will be a t degree polynomial and this c times A polynomial if I evaluated at if I evaluate it at α_i, this will give me the value v_i.

So, that means collectively the vector of new values $v_1, v_2, \dots, v_i, \dots, v_n$ constitutes t out of n secret sharing of $c \cdot s$; that means if a value s is already secret shared among the parties and if there is a publicly known constant c from the field, then to generate a secret sharing a t out of n secret sharing of the value $c \cdot s$, the parties need not have to interact. They can just multiply locally their respective shares of s with this constant c and that will give them their respective shares for the value $c \cdot s$

So, that is what we mean by linearity operate operation here. In the same way if the parties want to compute a secret sharing of $c + s$, where c again is a constant from the field; then what the parties have to do? They do not have to interact with each other; each party can just go and locally add the value c to its respective share of s. So, P_i can add c to its respective share of s, namely s_i and that this operation will result in a vector of values; that vector of values will now lie on a polynomial whose degree is t and whose ith point is the ith component in this vector.

And in the same way if I want to generate, if the parties want to generate a t out of n Shamir sharing of $s + s'$, where s and s prime are secret shared by instances of Shamir secret sharing; then again, the parties need not have to interact with each other, each party just must locally add its share of s and s prime.

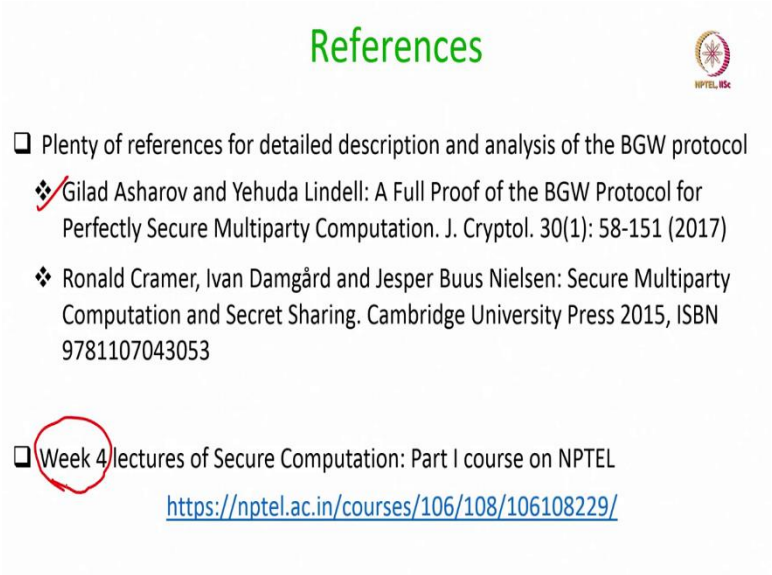
That will give the party its share of $s + s'$; because collectively now this vector $w_1, w_2, \dots, w_i, \dots, w_n$ lies on t degree polynomial, whose constant term is the sum of the constant terms of the A and B polynomial. The constant term of the A polynomial was s,

the constant term of the B polynomial was s'; that is why the constant term of the $A + B$ polynomial will be $s + s'$ and this $A + B$ polynomial when evaluated at α_i, will give you $s_i + s_i'$ which is nothing, but the ith component in this new vector.

So, that is what we mean by the linearity of Shamir secret sharing. So, the linearity property basically tells us that, if you have inputs which are secret shared and if you want to perform any linear function, a publicly known linear function of those secret shared inputs; then that linear function can be computed in a non interactively non interactive fashion in the sense that, the parties can apply the same linear function on their respective shares of the inputs of the function and apply the same linear function on the shares of the input, that will give them there is corresponding shares of the output of that function, output of the linear function.

So, this linearity of the Shamir secret sharing is a very powerful property, which takes care of maintaining the PGW gate invariant for the linear gates in the BGW protocol; of course, for the multiplication gates, we must do some more work, but we will come to that part later.

(Refer Slide Time: 34:21)



So, with that I end this lecture. So, these are the references used for today's lecture they are up.

Let us start sir.

Ok, with that I end this lecture. So, there are plenty of references available for the detailed description and analysis of the BGW protocol, my personal favorite is this first one. And as I said, we had the secure computation part I course where we discussed only the passive corruptions. So, whatever I have discussed in the today's lecture, you can find them in a more detailed fashion in the week 4 contents of the part I course which is available at the following URL.

Thank you.