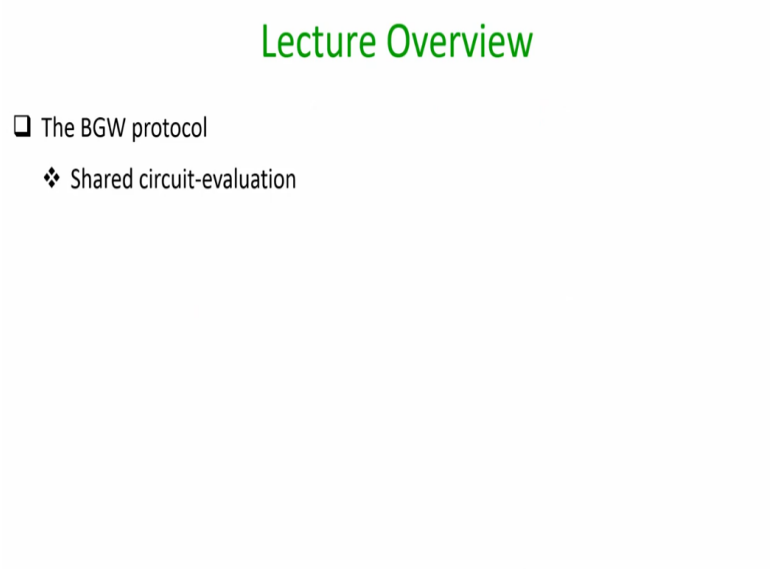
Secure Computation - Part I
Prof. Ashish Choudhury
Department of Computer Science
International Institute of Information Technology, Bangalore

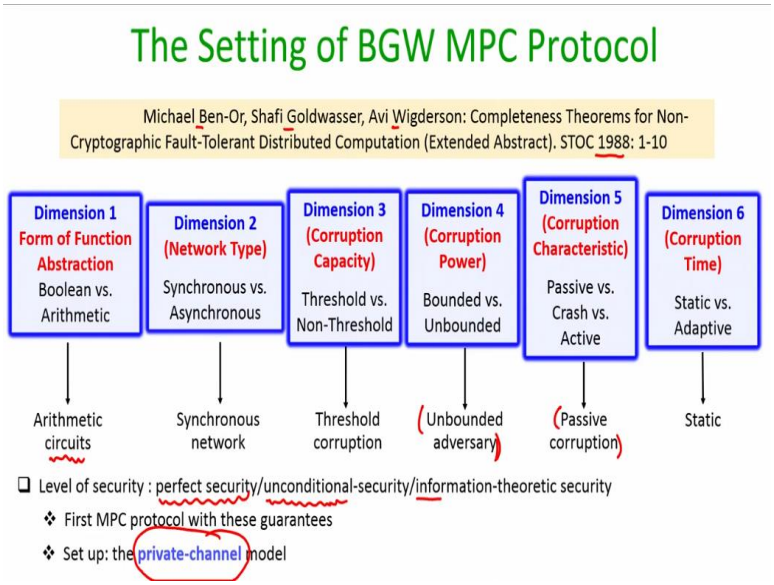
Module - 4
Lecture - 19
The BGW MPC Protocol

(Refer Slide Time: 00:34)



Hello everyone. Welcome to this lecture. The plan for this lecture is as follows: So, in this lecture, we will start discussing about BGW MPC protocol, which is one of the seminal results in the area of secure multi-party computation. And, we will introduce the concept of shared circuit-evaluation.

(Refer Slide Time: 00:52)



So, let us first try to understand the setting of BGW MPC protocol. So, this seminal result is attributed to Ben-Or, Goldwasser and Wigderson; that is why it is called the BGW protocol. This result was published in the year 1988. Remember that we can design MPC protocols in various dimensions. So, let us first try to understand the dimension in which the BGW MPC protocol is proposed.

It is a generic MPC protocol for any abstract function, where the function is represented by an arithmetic circuit over some finite field. It is designed assuming that the underlying network is a synchronous network. It tolerates a threshold adversary, where the corruption capability of the adversary is upper bounded by some publicly-known threshold t. The adversary here is assumed to be computationally unbounded. That is interesting here.

And even though the paper presents the protocol for both passive corruptions as well as malicious corruptions; since in this course, we are dealing with only passive corruptions; we will right now focus on passive corruptions. That means, we will assume that up to t parties can be passively corrupt, and they are computationally unbounded. In the next course, we will see the protocol to deal with t malicious corruptions.

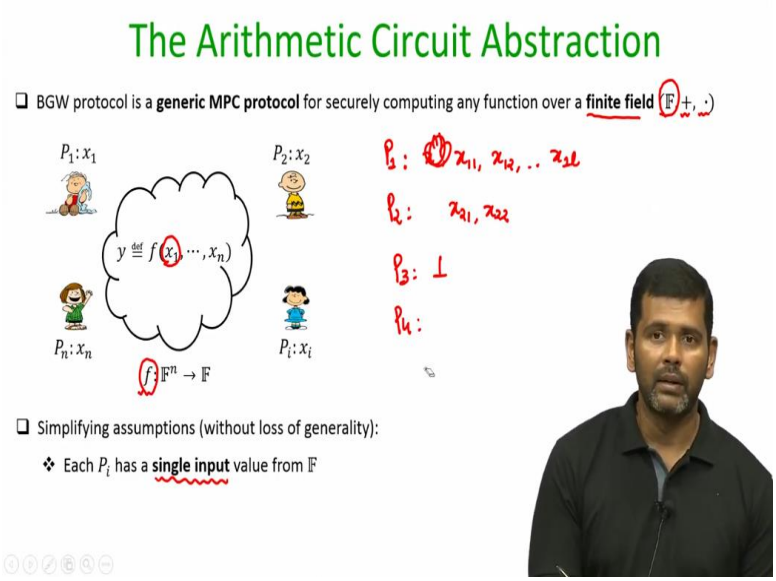
And for simplicity, we will assume that the adversary is static, which decides the set of corrupt parties *before* the beginning of the protocol itself. But the protocol can be proved to be secure even against an adaptive adversary, but for simplicity, we will stick to static corruptions. So, the level of security which is provided by the BGW MPC protocol is also called as perfect security, because we achieve security even against an adversary who is computationally unbounded.

So, the notion of the security achieved is also called sometimes as unconditional-security, because the security is not based on any computational hardness assumptions or any conditions. And it is also sometimes called as information-theoretic security. Historically, this is the first MPC protocol with these security guarantees, namely, perfect security or unconditional-security.

The protocol assumes the private-channel model, namely, it assumes that the underlying network is abstracted as a complete network, and there is a private channel between every pair

of parties. How do we instantiate such private-channel model? For that, you are referred to one of the earlier lectures for this course.

(Refer Slide Time: 04:04)



So, as I said that the circuit abstraction followed by the BGW protocol is that of arithmetic circuit. So, here we assume that the parties have some publicly-known function over some finite field. That finite field could be any finite field with an abstract plus an abstract dot operation. And the inputs of the parties are from the field. And the function output is also a field element.

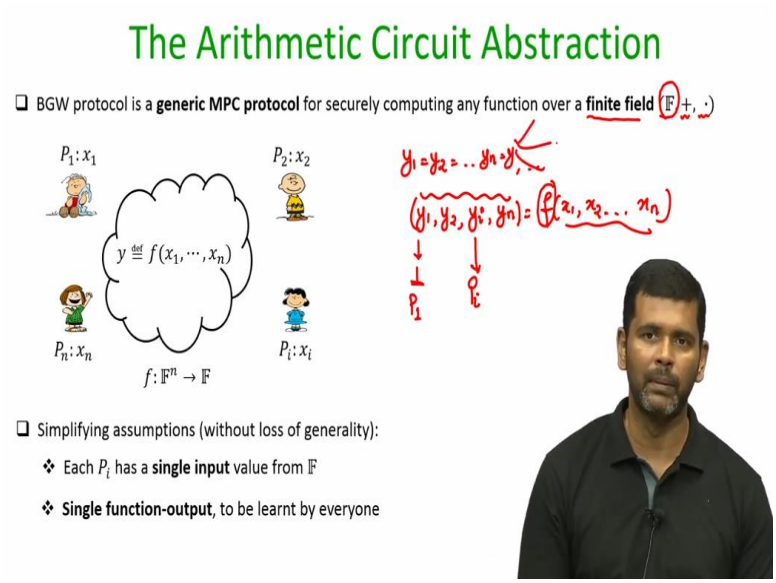
We will make several simplifying assumptions when discussing or presenting the BGW MPC protocol, but all these simplifying assumptions are without loss of generality. That means, even if these assumptions are not there, the protocol will work; but just for making the presentation easier, I am making these simplifying assumptions. The first simplifying assumption is that each party has a single field element as an input for the function.

So, I assume that party 1 has the input x_1, which is a private input, and a single field element which is the input of the party. Similarly, P_2 has a field element x_2, which is the private input for the function and so on. Of course, one can talk about a function where P_1 has several inputs possible for the function. Say P_1 has the inputs x_{11}, x_{12} and x_{13}. And P_2 might have 2 inputs, or m number of inputs; say x_{21}, x_{22}.

And P_3 might have no input. P_4 might have 3 inputs. So, I can design the BGW MPC protocol even to securely compute such kind of function, but that will require additional variables to

keep track of how many inputs for the function are coming from each party P_i. So, to avoid doing that, we make the simplifying assumption that each party has a single input for the function f, that is without loss of generality.

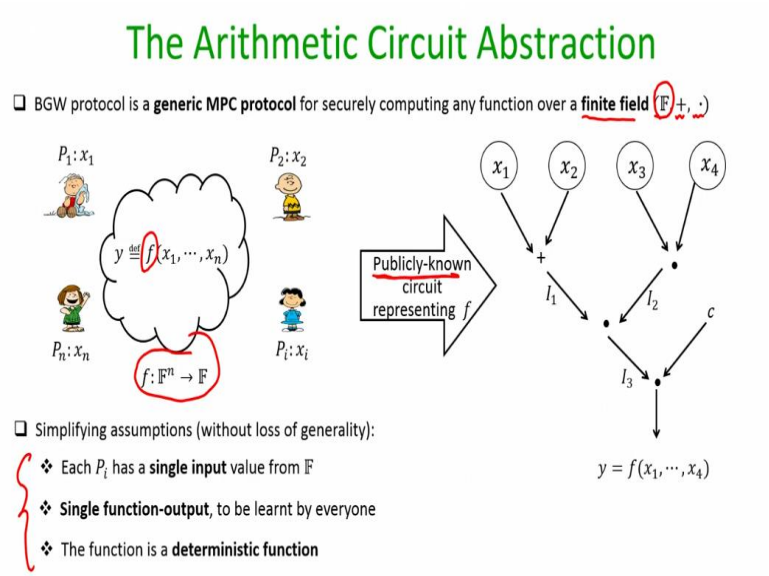
(Refer Slide Time: 06:27)



We also make the simplifying assumption that there is a single function output, which is supposed to be learnt by everyone, publicly it is supposed to be learnt. Again, there could be variations. So, for instance, it could be the case that my function itself is an n-ary output function, in the sense that it produces n outputs, based on the inputs of the parties, where the i^{th} output is supposed to be learnt only by P_i.

And these outputs could be different outputs, depending upon what is the nature of the function. In fact, some of the outputs could be a $\perp$. That means, there is no output for that specific party. That also could be the case. But I make the simplifying assumption that all these output values are a single field element which is going to be decided based on the function and the inputs; and this is supposed to be learnt by everyone, publicly it is supposed to be learnt by everyone. Again, this is the simplifying assumption, and this is without loss of generality.

(Refer Slide Time: 07:48)

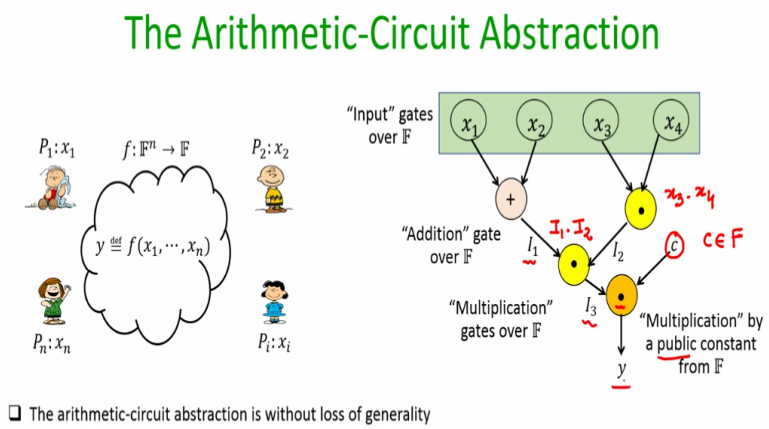


And the third simplifying assumption is that the function is a deterministic function. That means, once the values of the inputs $x_1, x_2, .., x_n$ are decided, then the function, the output is computed as a deterministic function, as per those inputs and the description of the function. There is no internal randomness which is used to compute the output of the function, namely, the output y. Again, this is without loss of generality.

We can easily modify the BGW MPC protocol to deal with the case where internal random field elements are generated when computing the output y, apart from the input values $x_1, .., x_n$. So, we will abstract out this function f with these simplifying assumptions as some publicly-known arithmetic circuit. That is important. The description of the arithmetic circuit will be publicly-known. So, for example, this could be a candidate function for 4 inputs.

Each input is coming from 1 party; so, input x_1 coming from first party and so on. And everyone will know that, okay, these are the various gates in the function f or the various gates in the circuit representing the function f.

(Refer Slide Time: 09:19)



Now, these gates can be categorised into various categories. So, to begin with, we will have what we call as input gates. They are called input gates because they are the input values which are supposed to be contributed by the respective parties. Then, in the circuit, we will have linear gates or what we call as addition gate over the field. So, this gate means, we want to add the values x_1 and x_2, and I am calling the result as I_1, intermediate value I_1.

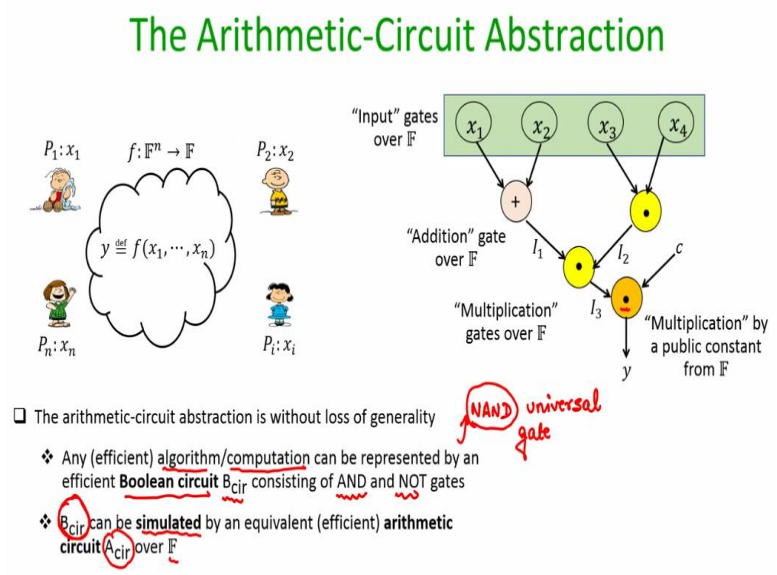
Or, the circuit could have multiplication gates, where the multiplications are performed over the finite field. So, we have 2 such multiplication gates present in this example circuit. The first multiplication gate is performing the product of x_3 and x_4 and the second multiplication gate is performing the product of the 2 intermediate results I_1 and I_2, and producing output I_3.

Now, you might be tempted to say that, okay, this is a multiplication gate as well; but this is not a multiplication gate. This is a multiplication by a publicly-known constant c, where c is publicly-known and an element of the finite field. So, this will be treated as a linear gate, because c times I_3 will be a linear function of I_3. It is equivalent to performing a linear computation of our I_3.

So, this is not a multiplication gate; but rather, this is a linear gate, because this is a multiplication by a *public constant*. And then, you have finally the output gate. So, now, very often, people ask the question that, is it a valid simplifying assumption to assume that the function can be abstracted by an arithmetic circuit? Can I represent any function, any computation, say an SQL query computation or performing some ML operations?

Can I abstract all such complex operations? Or, say the sorting algorithm; every algorithm, any kind of computation; can I abstract it as an arithmetic circuit over some finite field? And the answer is yes; this is without loss of generality.

(Refer Slide Time: 11:40)



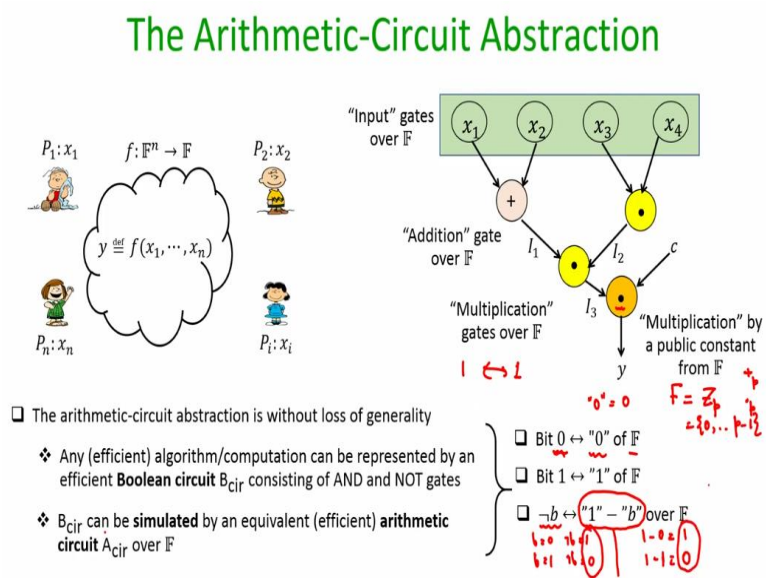
Because, if you take any efficient algorithm or any efficient computation; computation means, you are running some algorithm; it could be a graph algorithm; it could be a data structure algorithm; it could be any kind of algorithm. We can always represent it by an equivalent Boolean circuit, because, finally when you run your algorithm inside your computer, it is converted into binary code, and that binary code will be performing a sequence of Boolean operations inside your computer to get the result of the output of your algorithm.

That means, there will be an equivalent Boolean circuit which will be producing the same result as your algorithm or so called computation. Now, let us call that Boolean circuit as B_{cir}. And it is a well-known fact that NAND gates constitute universal gates. That means, any Boolean circuit which might have varieties of gates like XOR gates, OR gates, AND gates, NOT gates and so on, can be converted into an equivalent Boolean circuit which performs the same computation or produces the same result.

But now, the equivalent circuit will have only 2 types of gates, only AND gates and NOT gates, which together constitutes what we call as NAND gates. So, this is again a well-known fact. Now, my claim is that, that Boolean circuit consisting of just AND and NOT gates which is representing your underlying algorithm, can be also simulated by an equivalent arithmetic circuit over some finite field.

By simulated, I mean I can find a corresponding *arithmetic* circuit A_{cir}; and that circuit will be over the finite field where the values will be elements on the field, and all the operations will be only the $+$ and the $\cdot$ operations over the field. But that circuit A_{cir} will produce or will have the same effect as if the circuit B_{cir} is performing the computation.

(Refer Slide Time: 14:14)



How this is possible? Well, there are several ways to do this. One obvious way is the following: Whatever is your finite field that you have decided; say I have decided $\mathbb{F}$ to be the field $\mathbb{Z}_p$ consisting of the elements 0 to $p-1$; and my plus operation is $+_p$; and my multiplication operation is $\cdot_p$. Then, what I can do is the following:

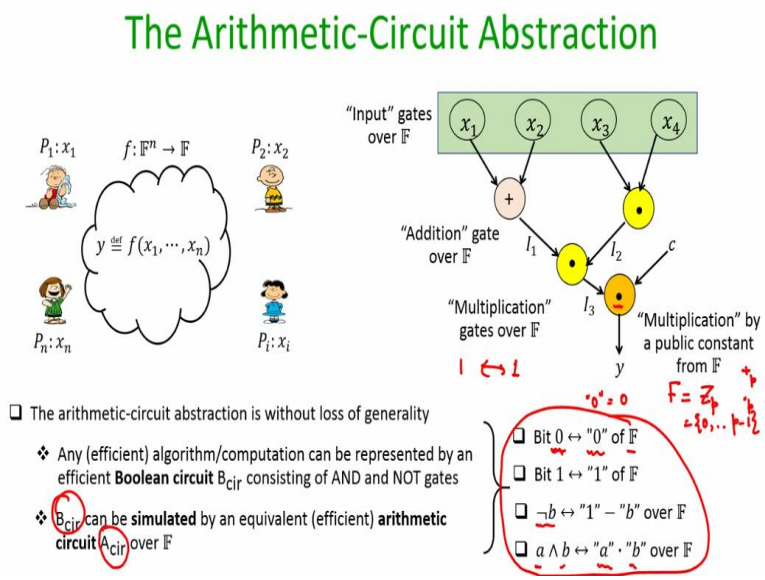
Wherever you have the bit 0 coming in the Boolean circuit, I decide a rule that all such occurrences of 0 will be now replaced by the element 0 of the *field*. So, if I take the field to be $\mathbb{Z}_p$, then the element 0 of the field is basically the element 0, the *integer* 0. But it could be any other abstract field, so, the bit 0 gets mapped to the 0 of the field, the bit 1 will be now mapped to the element 1 of the field.

Again, if I take $\mathbb{Z}_p$ to be my candidate field, then, what I am saying is that, wherever you have the bit 1 coming into picture, replace it with the *integer* 1. Wherever in the Boolean circuit you have an operation, where you are performing the negation of the bit b, that operation, you now replace by performing this arithmetic operation. Namely, whatever is the element 1; element 1 means the multiplicative identity element; minus the mapping of the bit b. That is, replace $\sim b$ with $1-b$.

So, again, if I take this $\mathbb{Z}_p$ as my candidate field, it is easy to see that this mapping works; because, if my $b = 0$, then $\sim b$ will be 1; if $b = 1$, negation of b will be 0. Now, let us see what will be the effect of performing the similar operation over the field, as per this mapping. The element 1 will be 1 itself, the integer 1 itself. So, $1 - b = 0$, that will produce you the result 1. And $1 - b = 1$ will produce the result 0 over the field.

So, now, you can see, you get the same answer which you would have got in the Boolean circuit. The same results, now you are getting by the equivalent arithmetic operations over the field.

(Refer Slide Time: 16:52)

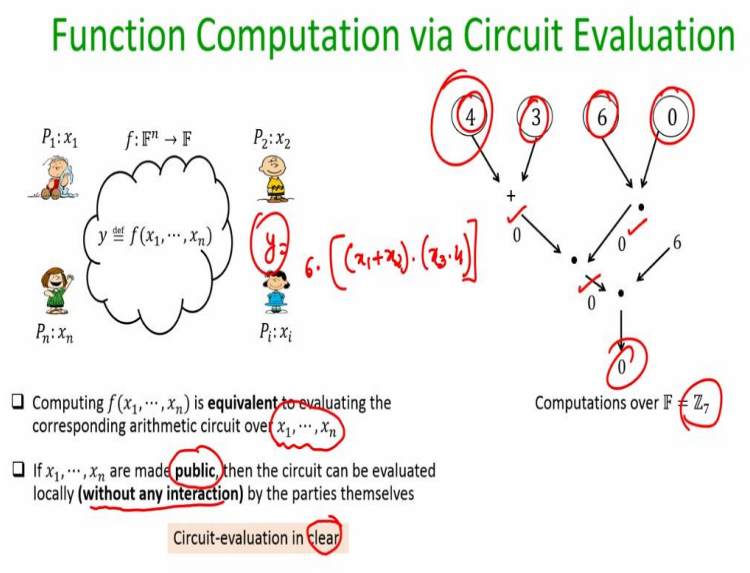


And now, what about the AND operation? Because, remember, in the Boolean circuit, I will have only 2 types of gates available, the NOT gate and the AND gate. So, if you have an AND gate, where the inputs are a and b, find an equivalent gate in the arithmetic circuit, where you perform the multiplication of the *mapped* a and the *mapped* b. Again, with the candidate $\mathbb{F}$ that we have considered $\mathbb{Z}_p$, this will work.

So, if your bits are a and b, then we know that the result of $a\ AND\ b$ will be 1 only when *both* the bits are 1. And the same will be the case if I consider the product of the field element a and the field element b. Because, if any of these two field elements a or b is 0, then the product of 0 field element with any other field element will give me the result 0. Only when both the field elements a and b are non-zero, I will get a non-zero answer.

So, that means, this mapping is always going to work; and by doing this mapping, I can always simulate the computations performed by the Boolean circuit by an equivalent arithmetic circuit, where the circuit just have values from the field and gates from the finite field.

(Refer Slide Time: 18:10)



So, from now onwards, we will assume that whenever we are talking about a function, I am basically talking about an equivalent arithmetic circuit. So, now, another thing to notice here is the following: If I say that if I want to compute the value of the function on the inputs $x_1, .., x_n$, that is equivalent to evaluating this arithmetic circuit over the inputs $x_1, .., x_n$. Because, if the values of $x_1, .., x_n$ are made public; suppose I ask every party that, okay, you make the values of your respective inputs public, announce it to everyone; then, everyone can compute the value of the function output.

Because, what they have to do is; they already know the description of the function and hence the corresponding arithmetic circuit; they can evaluate each and every gate of the circuit and obtain all the intermediate results and the final output. And this will not require any interaction once the inputs are made public. So, for instance, if I consider this example computation, where the function that we want to compute is $y = (x_1 + x_2) \cdot (x_3 + x_4) \cdot 6$; This is the overall function y.

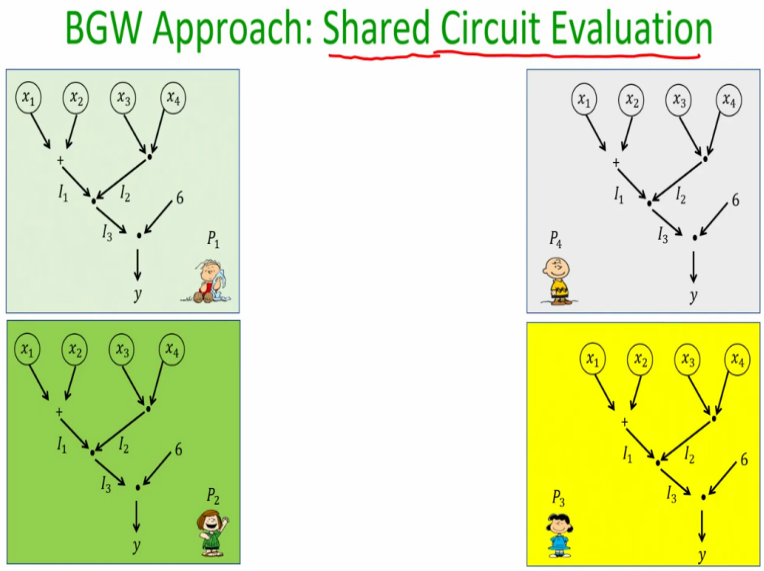
And now, if I take the candidate values of x_1, x_2, x_3 and x_4 to be 4, 3, 6 and 0; and say P_1 announces that, okay, my input is 4; P_2 announces that, my input is 3; P_3 announces that, my input is 6; and P_4 announces that, my input is 0. Then, after that, the parties need not have to

interact at all, and they can evaluate each and every gate, assuming that this is my field; and they will get the result of the function y.

And this is called circuit-evaluation in clear. Why in clear? Because you are knowing the value of the function input, and hence, you are computing the circuit in clear, knowing each and every intermediate value and the final output and also the input values. But, of course, this is not a secure solution; because, in this solution, even though this is a very nice solution, you do not require any interaction among the parties; just take the inputs of the parties and evaluate the circuit; but a whole security is breached here, because you are learning the inputs of every other party.

So, definitely, this is not the way we are going to compute functions *securely* in our MPC protocol, but we need to do something and evaluate the function or the equivalent circuit without revealing the inputs of the parties or the intermediate results.

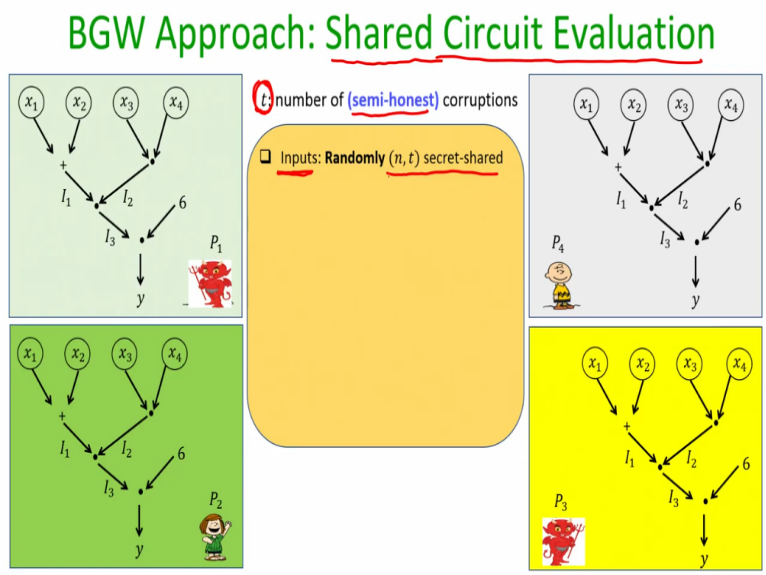
(Refer Slide Time: 21:21)



And this is where the very pioneering approach of BGW comes into picture. So, the BGW protocol introduced what we call as shared circuit-evaluation. This is very different from circuit-evaluation in clear, where the inputs are announced in public, and then every party just locally evaluates the circuit. In this approach, every party will be evaluating a copy of the circuit on some values, but the circuit will be evaluated in such a way that, if I consider any subset of t parties, from their viewpoint, they are basically evaluating the circuit on some garbage values; that is the idea here.

And this ensures that, if during the circuit-evaluation any set of t parties collude together, and exchange the version of the circuit they calculated among themselves, then that will not help them to reveal any information regarding the inputs of the parties; only the function output will be revealed. That is how the BGW approach performs circuit-evaluation.

(Refer Slide Time: 22:39)



So, as I said, the BGW protocol works under the assumption that t is the number of corruptions in the system. And we are talking about semi-honest corruptions here. By semi-honest corruptions, I stress that the parties will follow the protocol instructions, they will not deviate; that means, they will not send wrong values or they will not crash and so on. They will honestly perform the protocol instructions; but they are semi-honest in the sense that, once the protocol gets over, they will now try to infer something about the inputs of the other parties, based on whatever they have learnt in the protocol, which they are not supposed to do.

That is why they are semi-honest. So, the approach, the BGW approach behind this shared circuit-evaluation is as follows: To begin with, it will be ensured that the inputs for the function which are going to come from the respective parties, they are going to be made available to the parties in a secret-shared fashion, where the threshold of the sharing will be t, because t is the maximum number of corruptions in the system.

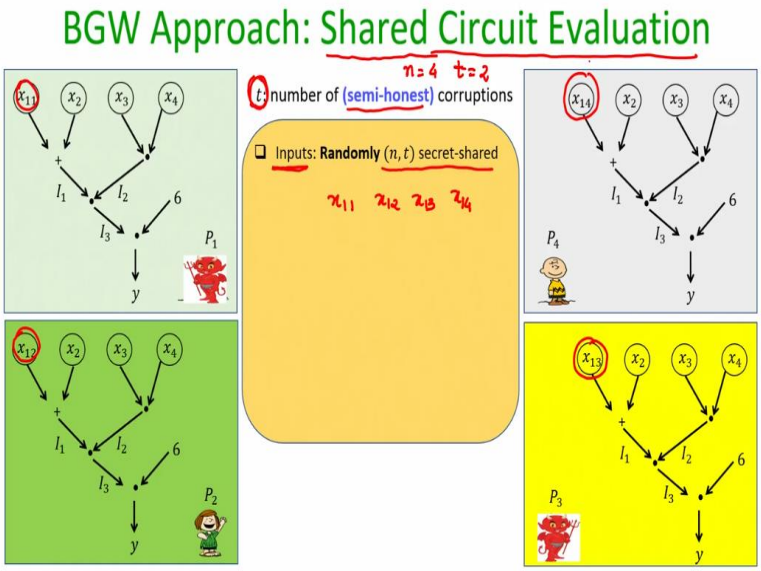
And t is a parameter which will be publicly-known. Remember, n is the number of parties that is publicly-known; t again is a parameter which basically denotes the maximum number of corruptions which can happen in the system. That means, this value t will be known, but who

exactly are those set of t corrupt parties, no one will know, when the protocol starts. It is only the adversary who knows that, okay, I am going to control these set of t parties.

So, to begin with, in the BGW shared circuit-evaluation, the inputs of the parties will be secret-shared, and they will be secret-shared in a random way; in the sense that, each party will have a single share for each input of the function; and together, the vector of n shares constitute a random (n, t) secret-sharing for that input. What does that mean? So, if I consider this input x_1; in circuit-evaluation in clear approach, P_1 would have announced publicly that this is the value of my input x_1.

We are not doing that. Instead, in the BGW approach, we are asking the party number P_1, okay, you are the owner of the input x_1, you create a secret-sharing for your input x_1 by running an instance of an (n, t) secret-sharing scheme. So, P_1 will act as the dealer, because it is the owner of the input x_1, and it will create shares of the input x_1.

(Refer Slide Time: 25:44)

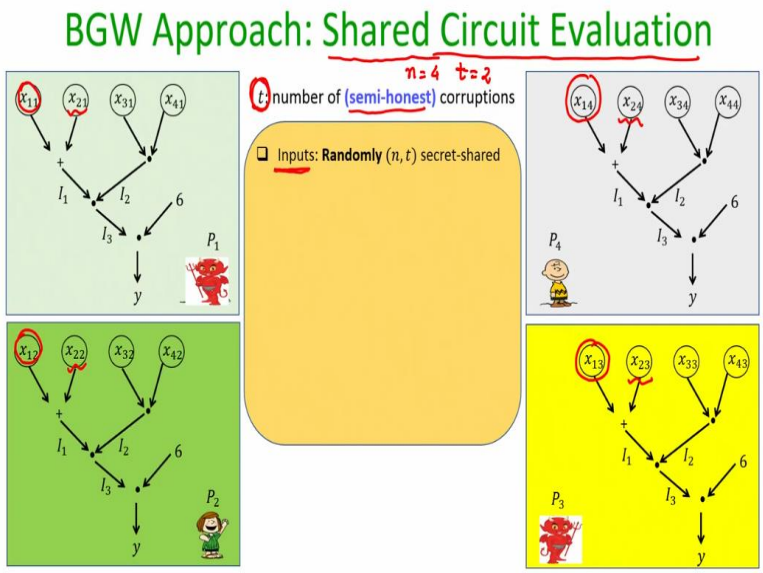


So, I am taking here the case where $n = 4$ and $t = 2$ for the sake of demonstration. So, it will create 4 random shares for its input x_1, which will have the property that any set of 2 shares among those 4 shares, leaks no information about the input x_1; but 3 or more number of shares are sufficient to recover back the input x_1. And now, it will do the following: It will distribute the shares of its input. I am calling those shares as x_{11}, x_{12}, x_{13} and x_{14}.

One share of the input x_1 will be given to each party. So, P_1 will have its own share for its own input x_1; P_2 will have its share x_{12}; P_3 will have its share of x_1, call it x_{13}; and P_4 will have its share for the input x_1, call it x_{14}. By doing this, it will be ensured that, if P_1 is not under the control of the adversary; well, in this example, I am taking P_1 to be under the control of the adversary; so, of course, it knows x_1, and of course all the 4 shares; because P_1 itself has distributed those 4 shares; and since adversary is instructing or controlling the P_1, it will know what are those 4 shares.

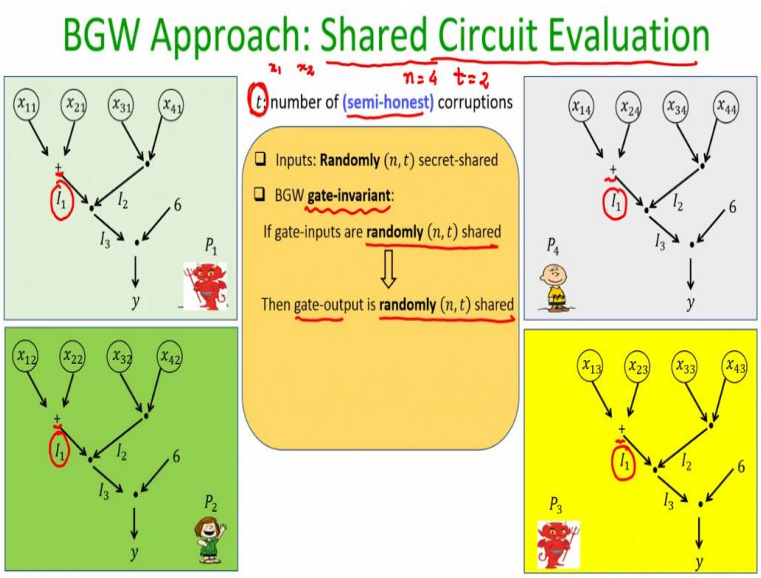
But consider another execution where P_1 is not under the control of the adversary; it is another set of 2 parties. Those 2 parties will get access to 2 shares of x_1, but those 2 shares will be *insufficient* to identify what exactly is the value of x_1. The same thing will be now done for the input x_2.

(Refer Slide Time: 27:32)



4 random shares x_{21}, x_{22}, x_{23} and x_{24} will be computed and distributed to the respective parties. It constitutes a vector of 4, 2 secret-sharing for x_2. Same will be done for x_3 and same will be done for x_4. This will complete the input stage. You now see that the inputs are no longer announced in public, but rather they are made available in a secret-shared fashion.

(Refer Slide Time: 28:05)

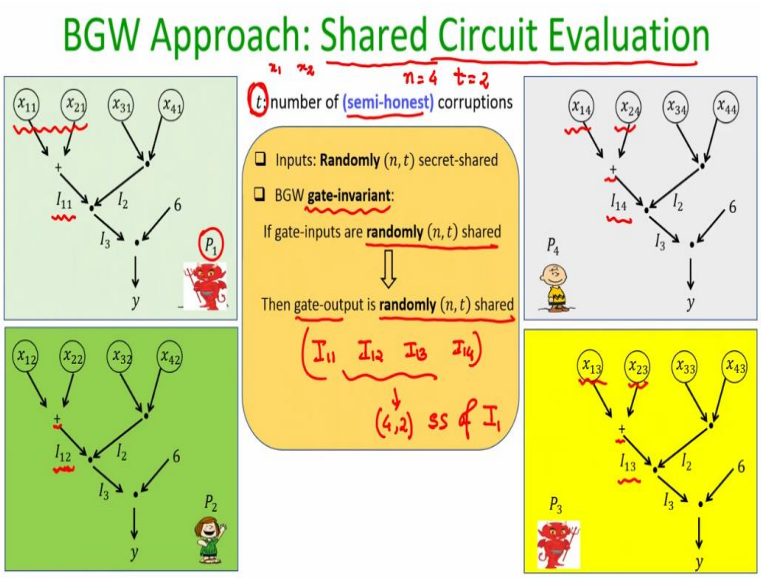


Now, comes the interesting part, the gate evaluations. And while performing the gate evaluation, the parties will try to maintain what we call as BGW gate-invariant. What does this invariant tries to maintain? The invariant that the parties try to maintain is the following: If the gate-inputs are randomly (n, t) secret-shared, then the parties try to ensure that even the gate-output is made available in a secret-shared fashion.

So, for instance, if the value of x_1 and x_2 would have been known in clear, everyone could have found the value of I_1, the first intermediate output, because they would have to just add the value of x_1 and x_2. But right now, no one knows the exact value of x_1 and no one knows the exact value of x_2. The gate-invariant tries to ensure that, if we consider this plus gate; and right now, every party has its own share for the inputs of this plus gate.

The inputs are x_1 and x_2. No one knows the value of x_1 and x_2, but each party has its own share of x_1 and own share of x_2. This BGW gate-invariant will try to ensure that, based on the shares of the inputs of this plus gate, each party is somehow able to compute a share of the intermediate gate-output.

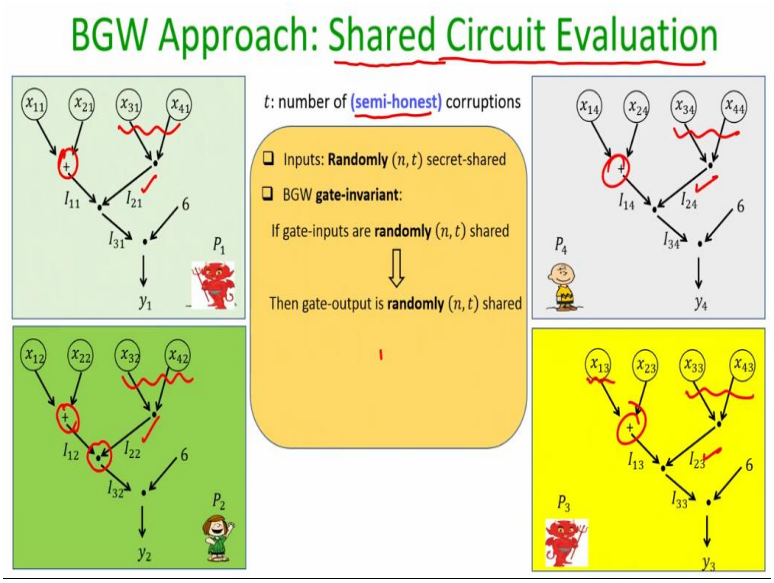
(Refer Slide Time: 29:44)



That means, P_1 will perform some computation on x_{11} and x_{21} to arrive at I_{11}. P_2 will try to perform some computation on x_{12} and x_{22} and arrive at this share I_{12}. P_3 will perform some computation on x_{13} and x_{23} and arrive at this share I_{13}. And P_4 will perform some computation on x_{14}, x_{24} and arrive at the share I_{14}. And together, this is I_{11}, I_{12}, I_{13} and I_{14} will constitute a vector of $(4,2)$ secret-sharing of the clear value I_1.

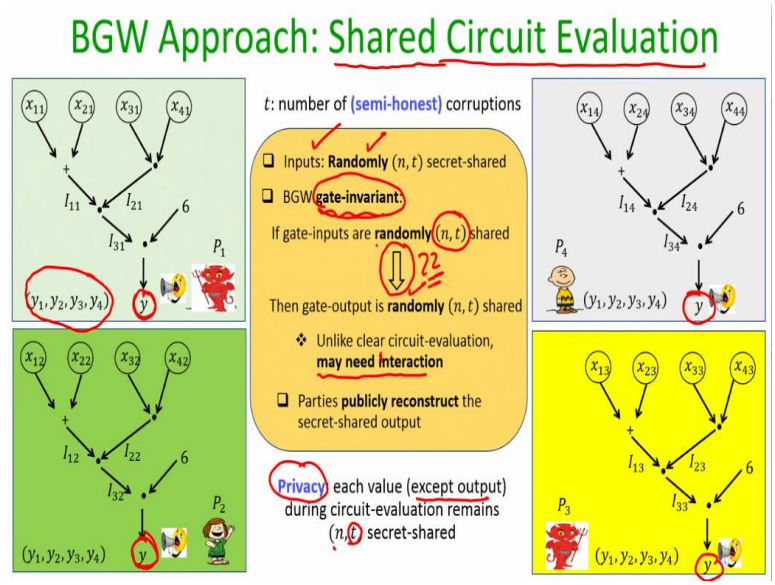
So, in essence, what this gate-invariant is trying to do is the following: Instead of evaluating the gates on clear inputs and obtaining the gate-output in clear, the gates are now evaluated on the shares of the gate-input and somehow parties are trying to obtain the shares of the gate-output; but in the process, no one learns what exactly was the exact value of the gate-input and what exactly was the exact value of the gate-output; because, everything is performed in a secret-shared fashion. And that is why this approach is called as the shared circuit-evaluation.

(Refer Slide Time: 31:18)



So, this plus gate will be evaluated like this by every party first. And then, similar thing, parties will do on the multiplication gate. Based on the shares of the multiplication gates, inputs of the multiplication gates, parties will perform some computation and obtain what we call as their respective shares of the intermediate value. And now, they will continue maintaining this invariant for all other gates, layer by layer by layer. Now, like this, once the entire circuit is evaluated in a secret-shared fashion;

(Refer Slide Time: 32:01)



By the way, maintaining this gate invariant might require interaction among the parties, because it may not always the case that just add the shares x_{11} and x_{21}, you get I_{11}, that may or may not be the case; or just multiply the shares x_{31}, x_{41} and get the share I_{21}, that may or may not be the case. How exactly this invariant is maintained, we will discuss that later. Right now, I am just explaining you the approach.

But looking ahead, we will see that maintaining this invariant may require *interaction* among the parties some time, unlike the circuit-evaluation in clear, where once the inputs are made public, no interaction is required among the parties. Now, once the function output is computed in a secret-shared fashion; that means, all the gates have been evaluated, and each party now has its own share for the final output; the parties publicly reconstruct that output.

And how they can reconstruct? Well, they can just announce their shares of this y value to each other. And right now, I am assuming semi-honest corruption; that means, even the bad parties, they will announce the correct value of the respective share of y. Now, once every party made their respective shares of y public, we will have the full vector of shares for the y value.

And now, the parties can apply the reconstruction algorithm of your secret-sharing scheme and reconstruct back the function output y. Now, why this shared circuit-evaluation will ensure the privacy property? Intuitively, what has happened in this entire shared circuit-evaluation? Each value, right from the input, all the way to the output, is not learnt by any party.

Of course, the party who is providing a particular input, it will know that, okay, this is my input and I have distributed these shares for my input. But for the inputs which are not known to a party, for those values, all together, the adversary sees t shares. And t shares, as per the (n, t) secret-sharing scheme, reveals no information about those values. And even for the intermediate results, assuming that this BGW gate-invariant is maintained securely.

That means, starting with the inputs of the gate which are secret-shared, we obtain the gate-output in a secret-shared fashion; and in the process, nothing additional about the gate-inputs and gate-output is revealed. What we are ensuring in this whole approach is that, each value except the output value remains secret-shared. And the degree of sharing throughout is ensured to be t.

And adversary, even though each party is evaluating its circuit, local copy of the circuit over some shares, adversary, if it controls t parties, it will see the evaluations of t parties. Through those t evaluations, basically for each value in the computation, it will get access to t shares.

But each value is going to be independently and randomly secret-shared with threshold t, and that does not help the adversary to learn any information about the intermediate results.

It is only the final output which is publicly reconstructed. But that is anyhow allowed to be learnt by every party, and that is not a privacy breach. And that is why this BGW approach ensures the privacy property. So, now the big question is, how exactly the inputs are secret-shared? And how exactly this gate-invariant is maintained? How can it be possible that without even knowing the gate-inputs, but every party being just given access to its share for the input values, somehow they magically obtain the shares of the output value? How exactly that gate-invariant is maintained? That is precisely is the whole crux of BGW MPC protocol.

(Refer Slide Time: 36:26)

So, these are the references which are used for today's lecture. There are several nice papers available for the detailed description and analysis of BGW protocol. These two are very nice resources, and you are referred to them. Thank you.