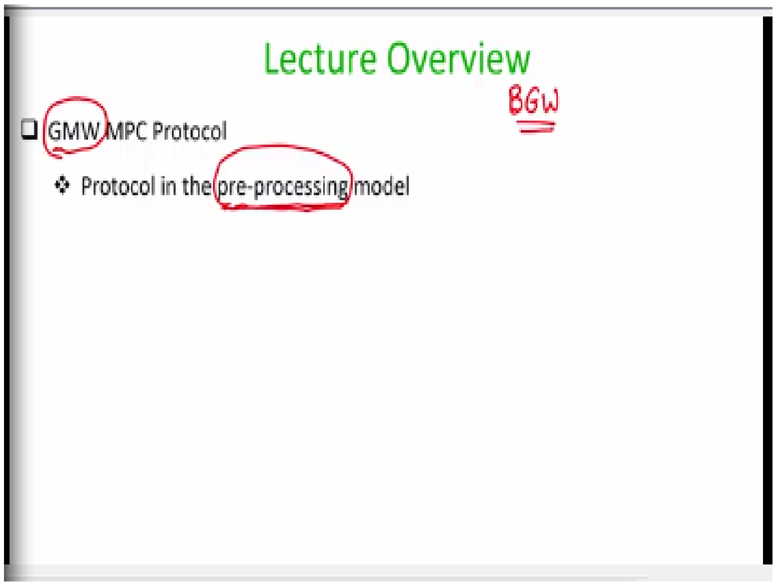
Secure Computation: Part I

Prof. Ashish Choudhury
Department of Computer Science Engineering
Indian Institute of Technology-Bengaluru

Lecture-36
GMW MPC protocol

(Refer Slide Time: 00:32)



Lecture Overview

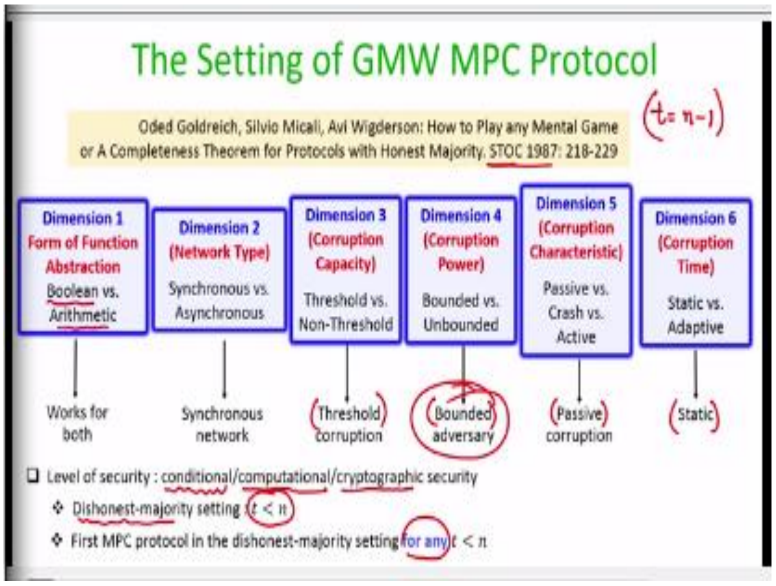
BGW

- GMW MPC Protocol
  - Protocol in the pre-processing model

Hello everyone, welcome to this lecture. So, in this lecture we will discuss the MPC protocol due to Goldreich, Micali and Wigderson, which is called the GMW MPC protocol. Like the perfectly secure MPC protocol due to Ben-Or, Goldwasser and Widgerson or the BGW MPC protocol, this GMW MPC protocol is again a very, very fundamental result in the literature of multi party computation. So, we will not be discussing the protocol in it is original form, we will study the protocol in the pre processing model.

Namely, we will assume that there is a pre processing phase, where some raw data is generated and using that the circuit is evaluated. The reason we are studying this variant of the GMW MPC protocol is that, these days most of the MPC protocols are designed in the pre processing model. This holds even for the GMW protocol, they assumed that we will first do the pre processing and then actually do the circuit evaluation.

(Refer Slide Time: 01:42)



The Setting of GMW MPC Protocol

$(t = n-1)$

Oded Goldreich, Silvio Micali, Avi Wigderson: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. STOC 1987: 218-229

Dimension 1 Form of Function Abstraction	Dimension 2 (Network Type)	Dimension 3 (Corruption Capacity)	Dimension 4 (Corruption Power)	Dimension 5 (Corruption Characteristic)	Dimension 6 (Corruption Time)
Boolean vs. Arithmetic	Synchronous vs. Asynchronous	Threshold vs. Non-Threshold	Bounded vs. Unbounded	Passive vs. Crash vs. Active	Static vs. Adaptive
Works for both	Synchronous network	Threshold corruption	Bounded adversary	Passive corruption	Static

- Level of security : conditional/computational/cryptographic security
  - Dishonest-majority setting $(t < n)$
  - First MPC protocol in the dishonest-majority setting for any $t < n$

So, this protocol was published in this paper in 1987 and let us first see the dimensions in which the protocol is proposed. The GMW MPC protocol is a generic MPC protocol. It tells you how to securely compute any function and a function is abstracted by a circuit and a circuit could be a Boolean circuit or it could be an arithmetic circuit.

So, the protocol works for both, this is unlike your BGW protocol where the circuit representation was an arithmetic circuit representation. The underlying network is assumed to be a synchronous network. That means the protocol will operate in a sequence of rounds, where in each round, the parties will process the messages that they got at the end of the previous round.

Then based on their own input and randomness and the messages they have received, they decide what to send for the current round and then they send those messages to the respective parties. And there will be strict bounds on the timeouts. That means if an expected message is not coming from a sender party, then the receiving party can conclude that the sender party was corrupt. And then it can decide either to ignore the messages from that sender party for the rest of the protocol execution or it can substitute some default value and proceed.

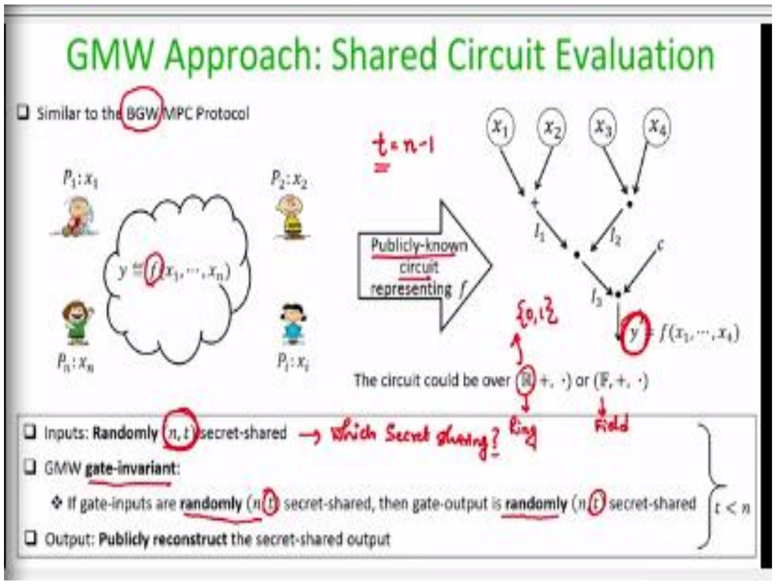
It assumes a threshold corruption, namely it assumes that up to t parties in during the protocol execution can get corrupt. The adversary is assumed to be a polynomial time adversary. And in this course, we will focus only on the passive corruption, namely the GMW protocol where the corruption type is passive. But the original protocol was proposed both to deal with the passive corruption as well as malicious or active corruption.

And for simplicity, we will consider static adversary. So, the notion of security which is achieved by the GMW MPC protocol, it is called by various names. It is sometimes called as conditional security - why conditional security? Because we are achieving security based on the condition or the assumption that our adversary is computationally bounded and hence it cannot solve the so called hard problems, intractable problems.

The notion of security is also called as computational because there are some computational hardness assumptions which we make to get the security. And those computational assumptions hold, because the adversary cannot solve those difficult problems. And sometimes the level of security is also called as cryptographic security. So, for the passive corruption, we will be considering the dishonest majority setting, where up to $n - 1$ parties can be under the control of the adversary.

So, in fact for the entire discussion of the GMW protocol, you can fix your t to be equal to $n - 1$, but the protocol will work for any $t < n$. But we will take the largest value of n, namely the maximum corruption which can be done by the adversary. Because that is the maximum damage which can adversary cause in the protocol.

(Refer Slide Time: 05:27)



GMW Approach: Shared Circuit Evaluation

- Similar to the BGW MPC Protocol

The circuit could be over $(\mathbb{R}, +, \cdot)$ or $(\mathbb{F}, +, \cdot)$

- Inputs: **Randomly** (n, t) secret-shared → Which Secret sharing?
- GMW **gate-invariant**:
  - If gate-inputs are **randomly** (n, t) secret-shared, then gate-output is **randomly** (n, t) secret-shared
- Output: **Publicly reconstruct** the secret-shared output

$t < n$

So, the approach of the GMW protocol is shared circuit evaluation similar to the BGW protocol. So, ideally it should be the other way around the BGW protocol approach is that of GMW because GMW was published earlier than BGW. But in this course we first studied the BGW protocol because we started with the highest possible level of security. So, we assume that the function which the parties want to securely compute is represented by some publicly known circuit.

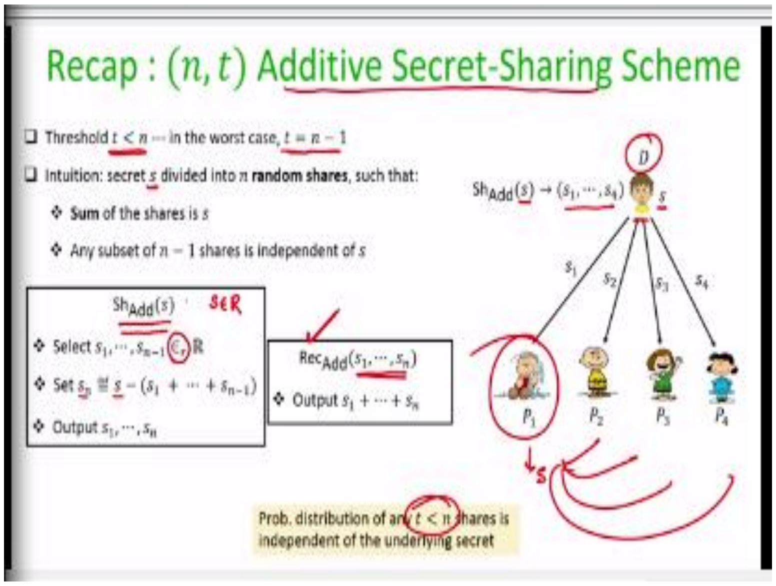
And that circuit could be either over a ring or it could be over a field. So, the GMW protocol will work irrespective of whether your circuit is over a ring or whether it is over a field. The approach of shared circuit evaluation that the GMW protocol follows is similar to the BGW approach. Namely, we will start with the following; the inputs of the function which are held by the respective parties will be randomly secret shared through a threshold secret sharing scheme where that threshold will be t.

And then we will maintain the gate invariant, where it will be ensured that if the gate inputs are randomly secret shared with threshold t. Then the protocol tries to ensure that the output of that gate also made available in a secret shared fashion, where the degree of sharing will be t and the shares will be constituting a vector of random shares. So, that means we start with a random secret sharing of the inputs and we end up with a random secret sharing of the gate output.

And in the process nothing additional about the gate inputs and a gate output is revealed. And then once we have the function output available in a secret shared fashion, the parties go and reconstruct it. So, that means the GMW protocol is based on the secret sharing approach. So, now the question is which secret sharing to use? We cannot necessarily use Shamir secret sharing because if the circuit is over a ring, say over a Boolean ring, where I have only 2 values 0 and 1.

Then we do not know how to do Shamir secret sharing because it requires at least n evaluation points. And more importantly our threshold t could be as large as $n-1$, then if we try to evaluate circuits over a field with threshold t being $n-1$ using polynomials of degree t, then we do not know how to do degree reduction and so on. So, that is why we cannot afford to use Shamir secret sharing scheme here, while secret sharing the values.

(Refer Slide Time: 08:40)



Recap : (n, t) Additive Secret-Sharing Scheme

- Threshold $t < n$ --- in the worst case, $t = n - 1$
- Intuition: secret s divided into n **random shares**, such that:
  - **Sum** of the shares is s
  - Any subset of $n-1$ shares is independent of s

$\mathsf{Sh}_{\mathsf{Add}}(s)$ $s \in R$
- Select $s_1, \cdots, s_{n-1} \in_r R$
- Set $s_n \stackrel{\text{def}}{=} s - (s_1 + \cdots + s_{n-1})$
- Output $s_1, \cdots, s_n$

$\mathsf{Rec}_{\mathsf{Add}}(s_1, \cdots, s_n)$
- Output $s_1 + \cdots + s_n$

Prob. distribution of any $t < n$ shares is independent of the underlying secret

$\mathsf{Sh}_{\mathsf{Add}}(s) \rightarrow (s_1, \cdots, s_4)$
D
s
s_1
s_2
s_3
s_4
P_1
P_2
P_3
P_4
s

So, which secret sharing will use? We will use additive secret sharing. So, let us quickly recap the (n, t) additive secret sharing scheme, where the threshold $t < n$ and as I said in the worst case t could be as large as $n-1$. And intuitively the way a value is shared in the secret sharing scheme is the following. If there is a value s which needs to be secret shared then it is divided randomly into n shares, it is randomly cut into n pieces.

This is done in such a way that if we take the sum of those n shares then we get back the value. But if we focus on any subset of $n-1$ shares, the probability distribution of those n$n-1$ shares will be independent of the secret s. That means those $n-1$ shares could be equi-probably the shares for any value from the secret space. So, here is the sharing protocol for the additive secret sharing scheme.

Assuming that all the operations are performed over a ring and your secret is also an element of the ring $\mathbb{R}$. So, the first $n-1$ shares they are picked randomly, uniformly at random from the ring. And then the last to share is set to be such that if you sum up the n pieces, then we get back the value s. Namely, we set $s_n = s - (s_1 + s_2 + \cdots + s_{n-1})$.

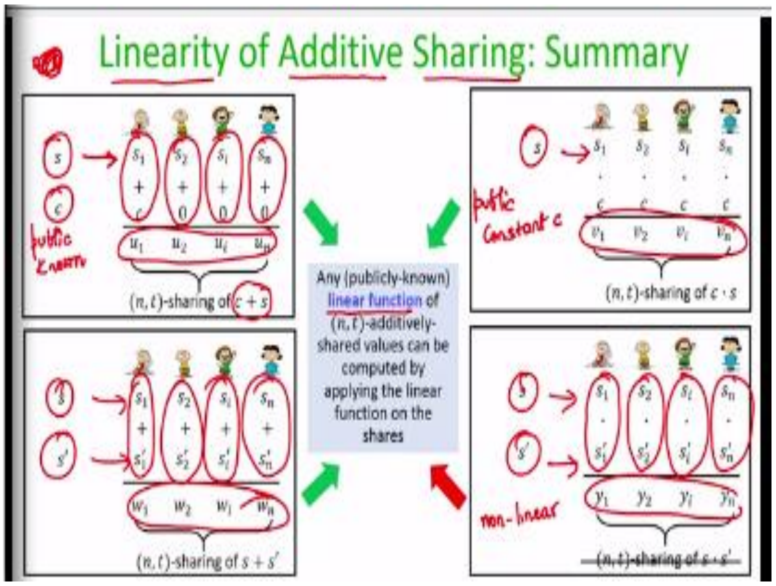
So, for instance if there is a dealer with a secret s, then to secret share this value among these 4 parties, it can compute these 4 shares randomly. So, it can pick the first 3 shares randomly and set the 4th share such that 4th piece along with the 3 random pieces that it has picked, sums up to the value s. And then it hands over the individual pieces to the individual shareholder. And if we want to reconstruct a value which has been secret shared, then we asked the parties to exchange the shares among themselves.

And if all the shares are available, then we just have to add them and that is a reconstructed value. Or if you want a communication efficient reconstruction protocol, then we can first let only say the first party or any designated party to reconstruct the value. For this we ask all the remaining shareholders to give their shares only to that designated shareholder. And that designated shareholder will now have all the n pieces.

So, it can run the reconstruction protocol and once it gets the value s, it can relay it to everyone else. And I will not be again going through the proof part, when we discuss additive secret sharing; there we prove that the secret sharing scheme is in fact perfectly secure. That means even if a set of t computationally unbounded, corrupt shareholders try reconstruct back the secret, they learn nothing about the underlying secret.

Because the secret sharing scheme has this nice property that you focus on any subset of t shares where $t < n$. Then the probability distribution of those t pieces which have been produced by this secret sharing scheme is independent of the actual value of the secret s.

(Refer Slide Time: 12:23)



Linearity of Additive Sharing: Summary

Any (publicly-known) linear function of (n, t)-additively-shared values can be computed by applying the linear function on the shares

s
c
public constant
$s_1 + c$, $s_2 + 0$, $s_i + 0$, $s_n + 0$
u_1 u_2 u_i u_n
(n, t)-sharing of $c + s$
public constant c
$s_1 \cdot c$, $s_2 \cdot c$, $s_i \cdot c$, $s_n \cdot c$
v_1 v_2 v_i v_n
(n, t)-sharing of $c \cdot s$
s'
$s_1 + s'_1$, $s_2 + s'_2$, $s_i + s'_i$, $s_n + s'_n$
w_1 w_2 w_i w_n
(n, t)-sharing of $s + s'$
non-linear
$s_1 \cdot s'_1$, $s_2 \cdot s'_2$, $s_i \cdot s'_i$, $s_n \cdot s'_n$
y_1 y_2 y_i y_n
~~(n, t)-sharing of $s \cdot s'$~~

Now since we are going to use the secret sharing scheme for securely evaluating the circuit, we will see what nice things we can do using these secret sharing scheme. Namely, we want to see whether this secret sharing scheme has the linearity property. And again we had seen earlier that this additive secret sharing scheme has this linearity property. Namely, suppose you have a linear function which is publicly known, and the inputs of that linear function is not available directly, but available in a secret shared fashion. And say those inputs are secret shared as per additive secret sharing scheme, then even the function output of that linear function can be computed in a secret shared fashion. And for that the parties do not need to interact with each other, it can be done non interactively. So, for instance if s is a value and say for demonstration I take $n = 4$.

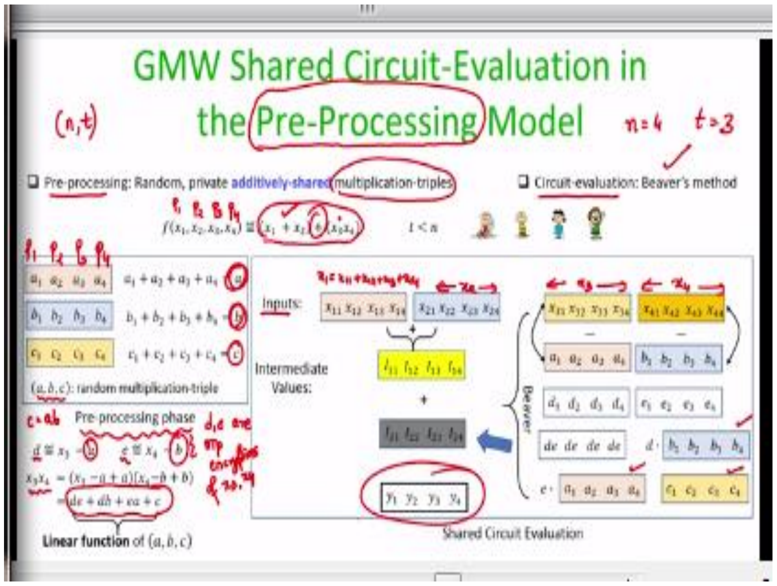
If s is a value which has been I am not taking $n = 4$, I am taking a general n. So, if s is a value which has been secret shared additively with parties holding the shares $s_1, s_2, .., s_i, .., s_n$ respectively. And if c is publicly known constant and if we now ask only say one of the designated parties to add c to it is share of s. And all other parties other than that designated party to just add 0 to their respective shares of s.

Then the resultant vector of values will constitute an additive secret sharing scheme on additive secret sharing of $c + s$. And this vector of shares for $c + s$ will be random, if the vector of shares for the s was random. In this whole process nothing about s is revealed because no interaction happens among the parties. In the same way suppose there is a value s and another value s' which has been secret shared.

And now, if we ask individual parties to just add up locally their respective shares of s and s', then the resultant vector of values that they obtain constitute a additive secret sharing of $s + s'$. And in this whole process nothing about s or s' is revealed. And again, if the vector of shares for s or the vector of shares for s' was random, then indeed $s + s'$ is also randomly secret shared.

Same way if parties take a public constant c and each party locally multiplies this constant c with it is respective share of the value s, then basically we obtained the value $c \cdot s$ in a secret shared fashion. But unfortunately, this additive secret sharing does not allow to do nonlinear operations in a non interactive way. Because if there is a value s and if there is a value s' which has been secret shared. And if we just asked the parties to locally multiply their respective shares of $s \cdot s'$, then the resultant vector of shares *does not* constitute an additive sharing of $s \cdot s'$.

(Refer Slide Time: 16:03)


GMW Shared Circuit-Evaluation in the Pre-Processing Model
(n,t)
n=4 t=3
Pre-processing: Random, private additively-shared multiplication-triples
Circuit-evaluation: Beaver's method
$f(x_1,x_2,x_3,x_4) \equiv (x_1+x_2) \cdot (x_3 x_4)$
$t < n$
$a_1+a_2+a_3+a_4 = a$
$b_1+b_2+b_3+b_4 = b$
$c_1+c_2+c_3+c_4 = c$
(a,b,c): random multiplication-triple
c = ab
Pre-processing phase
$d \equiv x_3 - a$
$e \equiv x_4 - b$
$x_3 x_4 = (x_3 - a + a)(x_4 - b + b)$
$= de + db + ea + c$
Linear function of (a,b,c)
Inputs:
Intermediate Values:
Beaver
Shared Circuit Evaluation

So, that is a recap of the additive secret sharing. And in the GMW shared circuit evaluation, we use the additive secret sharing. So, as I said I will not be discussing the GMW protocol in its original form, we will discuss it in the pre processing model. Assuming that the protocol is divided into a pre processing phase, where the parties generate circuit independent and function independent raw data which is now later going to be used while performing the actual circuit evaluation.

In the pre processing phase, we basically generate multiplication triples, because during the circuit evaluation the parties are going to use the Beaver's multiplication method. So, the multiplication triples which the parties will generate in the pre processing phase will be (n, t) secret shared same as was the case for the BGW protocol. But now instead of Shamir secret sharing scheme, the secret sharing that we will be following here is the additive secret sharing.

So, let me demonstrate this GMW protocol in the pre processing model, assuming that we have n equal to say 4 and say t = 3 and say the parties want to securely compute this function. So, how many multiplication gates are there in this function? There is one multiplication gate, so that is why in the pre processing phase, the parties will generate one random multiplication triple, that means a, b and c will random elements from the ring such that $c = a \cdot b$.

And no one will be knowing the value of a, no one will be knowing the value of b; no one will be knowing the value of c. Because a, b and c will be available in a secret shared fashion (n, t) additive secret shared fashion, namely each party will hold a share of a, share of b, a share of c. Now, assuming that there is a way to generate this pre processing data, we will focus later how to generate this pre processing data.

But assuming for the moment that such a pre processing has been already done, then the shared circuit evaluation as per the GMW protocol happens as follows. So, we start with the inputs, the inputs are x_1, x_2, x_3, x_4. So, each party who owns that input, so again for simplicity assume that P_1 owns x_1, P_2 owns x_2, P_3 owns x_3 and P_4 owns x_4. So, whoever owns the respective input of the function it acts as a dealer for the additive secret sharing scheme and secret share the respective inputs.

So, x_1 will be secret shared by P_1, namely it will be split into 4 random pieces summing up to x_1. The 1st party obtaining x_{11}, 2nd party obtaining x_{12}, 3rd party obtaining x_{13}, 4th party obtaining x_{14}. So, $x_1 = x_{11} + x_{12} + x_{13} + x_{14}$, similarly x_2 is secret shared by P_2, x_3 is secret shared by P_3,

x_4 is secret shared by P_4. Now, the party start evaluating the circuit because the inputs are ready now in a secret shared fashion, they are available, so they first take the addition gate.

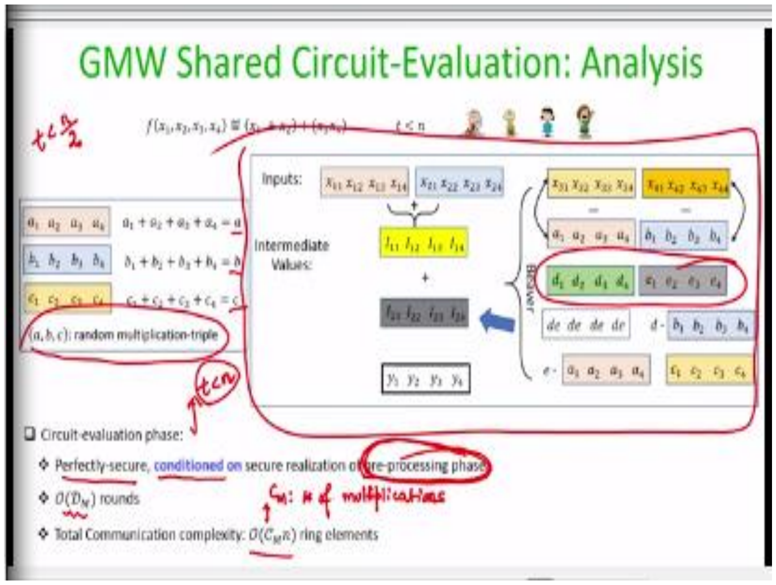
Addition is a linear operation, so that gate can be evaluated as per the GMW invariant. Each party just have to add it is respective shares of x_1 and x_2. Now we come to this multiplication gate, namely $x_3 \cdot x_4$ has to be evaluated in a secret shared fashion. And here we again resort to Beaver's method which helps us to express the result of $x_3 \cdot x_4$ as a linear function of a, b, c provided the values d and e are publicly available.

So, d, e are OTP encryptions of x_3 and x_4 using the values a and b as the parts. And how do we get these OTP encryptions in the public domain? Well, we first compute a secret shares of the OTP encryptions, that can be done by just to asking the parties to respectively subtract their local shares of x_3 and a and x_4 and b and then publicly reconstruct the OTP encryptions. That means for evaluating the product of $x_3 \cdot x_4$ in a secret shared fashion, we need to reconstruct 2 OTP encryptions in the public domain.

And reconstructing this OTP encryption in the public domain will not reveal any information about x_3 and x_4. Because a was randomly chosen, b was randomly chosen and both of them are secret shared, no one knows the value of a and no one knows the value of b. Now once the value of d and e are made public, we have to compute this linear function which will give us $x_3 \cdot x_4$.

But this linear function will be computed in a secret shared fashion because the inputs of this linear function, namely a, b and c are available in a secret shared fashion. So, the parties just compute the same linear function in a secret shared fashion and that will give them a secret shared representation for$x_3 \cdot x_4$. Now we have this plus gate in the circuit which can be computed or evaluated locally. And now once the function output y is ready, we go and reconstruct it.

(Refer Slide Time: 22:37)


GMW Shared Circuit-Evaluation: Analysis

- Circuit-evaluation phase:
  - Perfectly-secure, conditioned on secure realization of pre-processing phase
  - $O(D_M)$ rounds
  - Total Communication complexity: $O(C_M n)$ ring elements

$t < \frac{n}{2}$
$f(x_1, x_2, x_3, x_4) \stackrel{\text{def}}{=} (x_1 + x_2) + (x_3 x_4)$
$t < n$
Inputs:
Intermediate Values:
Beaver
$a_1 + a_2 + a_3 + a_4 = a$
$b_1 + b_2 + b_3 + b_4 = b$
$c_1 + c_2 + c_3 + c_4 = c$
(a, b, c): random multiplication-triple
$t<n$
C_M: # of multiplications

So, it is a very simple protocol assuming that this pre processing has been done. Now let us try to analyze this whole GMW shared circuit evaluation process. It turns out that this whole circuit evaluation protocol, this part is perfectly secure. That means if during the circuit evaluation phase, even if the t parties which are under the control of the adversary are computationally unbounded, the security is achieved.

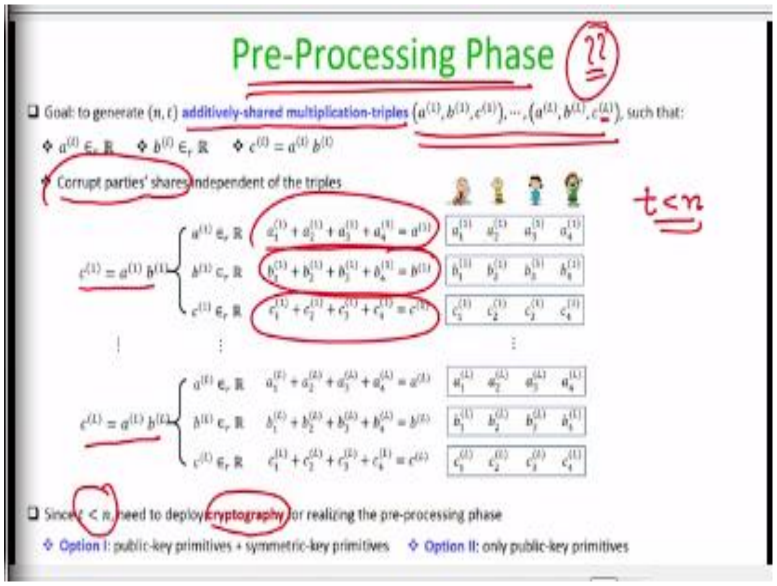
That means we get perfect security conditioned on the fact that you have a perfectly secure way of realizing or implementing the pre processing phase. That means if someone gives me a guarantee, that ok, you do not worry about the pre processing phase, we will ensure somehow that whatever you require from the pre processing phase that will be done, we will give you secret shared multiplication triples where no one knows the values of those multiplication triples, and every party just has a share of those multiplication triples - this arrangement has been done. Conditioned on this arrangement, the circuit evaluation part of the GMW protocol is perfectly secure even if $t < n$ and that is interesting. And say this is no way violating our requirement of $t < \frac{n}{2}$ bound for perfect security.

I am telling you that if you give me this pre processing phase, some way of implementing instantiating this pre processing phase with perfect security, then I can give you a method of circuit evaluation even tolerating $t < n$ corruptions. How many rounds will be required in the circuit evaluation phase? Again, every time there is a layer of multiplication gates, the parties have to interact because they have to publicly reconstruct the OTP encryptions of the multiplication gate inputs. So, that is why the number of rounds that are required here is proportional to the multiplicative depth of the circuit.

And per multiplication gate, 2 OTP encryptions have to be publicly reconstructed. Assuming that 1 public reconstruction takes a communication of order n ring elements, we have C_m number of multiplication gates. So, overall for evaluating all the multiplication gates in the circuit, these many ring elements have to be communicated. So, now everything boils down to how do you get this pre processing phase.

Because if you give me this pre processing phase, a secure way of implementing the pre processing phase, then the circuit evaluation is basically the same as your BGW protocol. Except that now the sharing semantic is additive sharing instead of polynomial based Shamir secret sharing. Namely, linear gates will be evaluated locally; multiplication gates will be evaluated using Beaver's method. And Beaver's method is independent of whether your secret sharing is additive or whether it is Shamir. Because it just requires to compute publicly the OTP encryptions of your multiplication gate inputs and it requires your underlying secret sharing scheme to be linear.

(Refer Slide Time: 26:22)



Pre-Processing Phase ??

- Goal: to generate (n, t) additively-shared multiplication-triples $(a^{(1)}, b^{(1)}, c^{(1)}), \cdots, (a^{(L)}, b^{(L)}, c^{(L)})$, such that:
  - $a^{(l)} \in_r \mathbb{R}$
  - $b^{(l)} \in_r \mathbb{R}$
  - $c^{(l)} = a^{(l)} b^{(l)}$
  - Corrupt parties' shares independent of the triples

$t < n$

$$c^{(1)} = a^{(1)} b^{(1)} \begin{cases} a^{(1)} \in_r \mathbb{R} & a_1^{(1)} + a_2^{(1)} + a_3^{(1)} + a_4^{(1)} = a^{(1)} \quad \boxed{a_1^{(1)} \; a_2^{(1)} \; a_3^{(1)} \; a_4^{(1)}} \\ b^{(1)} \in_r \mathbb{R} & b_1^{(1)} + b_2^{(1)} + b_3^{(1)} + b_4^{(1)} = b^{(1)} \quad \boxed{b_1^{(1)} \; b_2^{(1)} \; b_3^{(1)} \; b_4^{(1)}} \\ c^{(1)} \in_r \mathbb{R} & c_1^{(1)} + c_2^{(1)} + c_3^{(1)} + c_4^{(1)} = c^{(1)} \quad \boxed{c_1^{(1)} \; c_2^{(1)} \; c_3^{(1)} \; c_4^{(1)}} \end{cases}$$

$$\vdots$$

$$c^{(L)} = a^{(L)} b^{(L)} \begin{cases} a^{(L)} \in_r \mathbb{R} & a_1^{(L)} + a_2^{(L)} + a_3^{(L)} + a_4^{(L)} = a^{(L)} \quad \boxed{a_1^{(L)} \; a_2^{(L)} \; a_3^{(L)} \; a_4^{(L)}} \\ b^{(L)} \in_r \mathbb{R} & b_1^{(L)} + b_2^{(L)} + b_3^{(L)} + b_4^{(L)} = b^{(L)} \quad \boxed{b_1^{(L)} \; b_2^{(L)} \; b_3^{(L)} \; b_4^{(L)}} \\ c^{(L)} \in_r \mathbb{R} & c_1^{(L)} + c_2^{(L)} + c_3^{(L)} + c_4^{(L)} = c^{(L)} \quad \boxed{c_1^{(L)} \; c_2^{(L)} \; c_3^{(L)} \; c_4^{(L)}} \end{cases}$$

- Since $t < n$, need to deploy cryptography for realizing the pre-processing phase
  - Option I: public-key primitives + symmetric-key primitives
  - Option II: only public-key primitives

So, now let us focus on how do we get this pre processing phase? What exactly we require from the pre processing phase? We require to generate L number of random additively shared multiplication triples. Namely, we require triples of the form $(a_1, b_1, c_1), (a_2, b_2, c_2), \ldots., (a_L, b_L, c_L)$, where the c components are the product of the a and b components. All the triples are randomly chosen from the ring, no one knows the value of those triples and each of the triples is additively shared.

They are shared randomly in such a way that corrupt parties' shares, namely, any subset of t shares, is independent of the actual value of the multiplication triples. We then have to implement, realize this pre processing phase, tolerating up to $t < n$corruptions because that is a resilience bound for the entire MPC protocol. The entire MPC protocol consists of your pre processing phase as well as the circuit evaluation phase.

For circuit evaluation, we know that even if adversary is computationally unbounded, we can tolerate it. But it turns out that when we want to implement the pre processing phase and that too with$t < n$, we cannot do it in a perfectly secure way. We have to deploy cryptographic tools and that is why the overall GMW protocol becomes cryptographically secure or computationally secure.

Otherwise the circuit evaluation part of the GMW protocol is perfectly secure. It is only the pre processing phase where to generate this shared random multiplication triples, we need to resort 2 cryptographic primitives. And since we have to use cryptography, there are 2 options to generate this data structure that we require from the pre processing phase. Option 1 will be that we use both public key as well as symmetric key primitives, namely a combination of both of them, option 2 will be that we use just public key primitives.

So, we will now shift our attention on how to get this pre processing phase. Because once the pre processing phase is done, the circuit evaluation phase is very simple. So, with that I end this lecture, thank you.