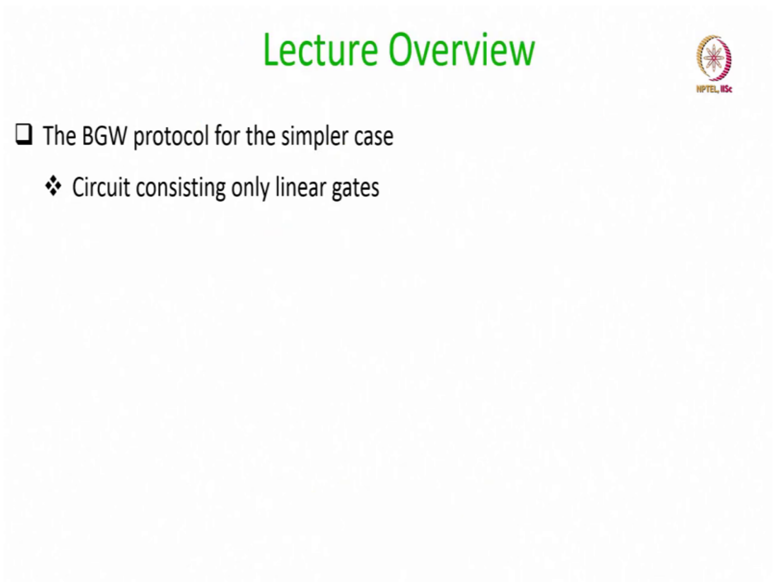
Secure Computation: Part II

Prof. Ashish Choudhury
Department of Computer Science and Engineering
Indian Institute of Science, Bengaluru

Lecture - 44
The BGW MPC Protocol for Linear Functions

(Refer Slide Time: 00:26)

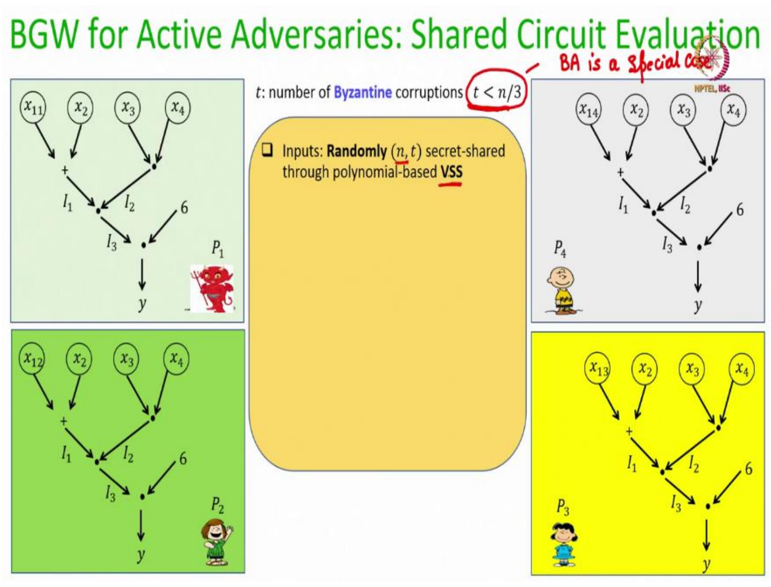


Lecture Overview

- ❑ The BGW protocol for the simpler case
  - ❖ Circuit consisting only linear gates

Hello everyone, welcome to this lecture. So, in this lecture we will now start from this lecture onwards we start discussing about the BGW Protocol for perfectly secure MPC and we start first with the simpler case namely, we consider the case when the underlying function is represented by a circuit consisting of only linear gates.

(Refer Slide Time: 00:48)


BGW for Active Adversaries: Shared Circuit Evaluation

BA is a Special case

t: number of Byzantine corruptions $t < n/3$

- Inputs: **Randomly** (n, t) secret-shared through polynomial-based **VSS**

NPTEL, IISc
x_{11} x_2 x_3 x_4
x_{14} x_2 x_3 x_4
x_{12} x_2 x_3 x_4
x_{13} x_2 x_3 x_4
+
l_1
l_2
l_3
6
y
P_1
P_2
P_3
P_4

So, let me quickly go through the idea behind the BGW protocol namely the broad principle of shared circuit evaluation. So, imagine you have some circuit which is publicly available the circuit is over some finite field and in the system there could be up to t number of Byzantine corruptions malicious corruptions where $t < \frac{n}{3}$. Now, I stress that this condition $t < \frac{n}{3}$ is a necessary condition for any perfectly secure multiparty computation and we can very easily prove this.

What we can say here is that the Byzantine agreement problem is a special case of MPC and we have already we had already proved that for perfectly secure Byzantine agreement the condition $t < \frac{n}{3}$ is necessary that automatically shows that for generic functions for computing any generic function or arbitrary function abstract function with perfect security we require the condition necessary condition $t < \frac{n}{3}$.

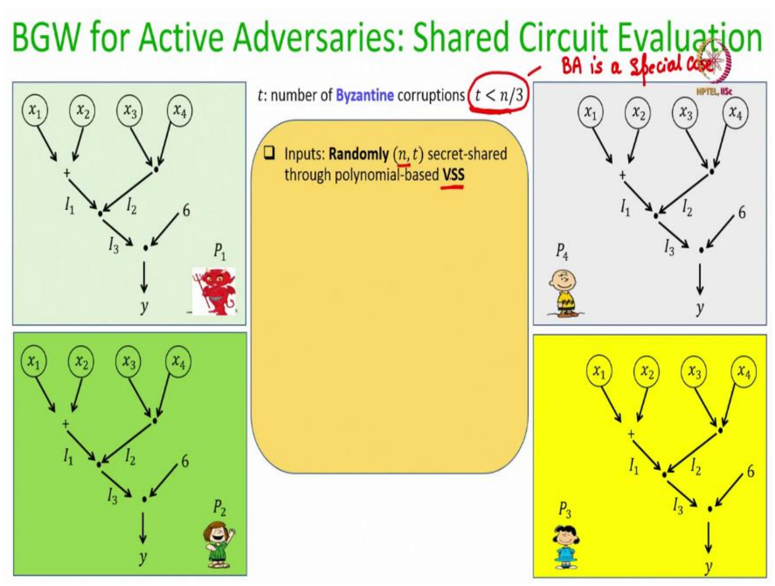
So, there will be various stages in the BGW protocol depending upon what gates you are evaluating. So, we start with the input stage where all the inputs for the function are made available in a secret shared fashion using a (n, t) secret sharing scheme based on polynomial based on polynomials. And for doing that we will used polynomial based VSS the VSS schemes which we had discussed till now.

So, we can use the VSS scheme with $t < \frac{n}{3}$ which we had discussed earlier and now each party is going to do the following say for instance we take the party P_1 it owns the input

x_1. So, what it can do is it can act as a dealer and take x_1 as its secret and run the sharing phase protocol for secret sharing the value x_1 in an (n, t) secret shared fashion.

Due to which at the end of the sharing phase protocol every party will have a share for the value x_1 and it will be verified during the sharing phase protocol that indeed party P_1 has secret shared sum x_1 using a t degree polynomial that verifiability comes because of the strong commitment and the correctness property of the verifiable secret sharing scheme. So, even if P_1 is corrupt even if it is a corrupted dealer, it cannot distribute arbitrary shares for its input x_1.

(Refer Slide Time: 03:44)



BGW for Active Adversaries: Shared Circuit Evaluation

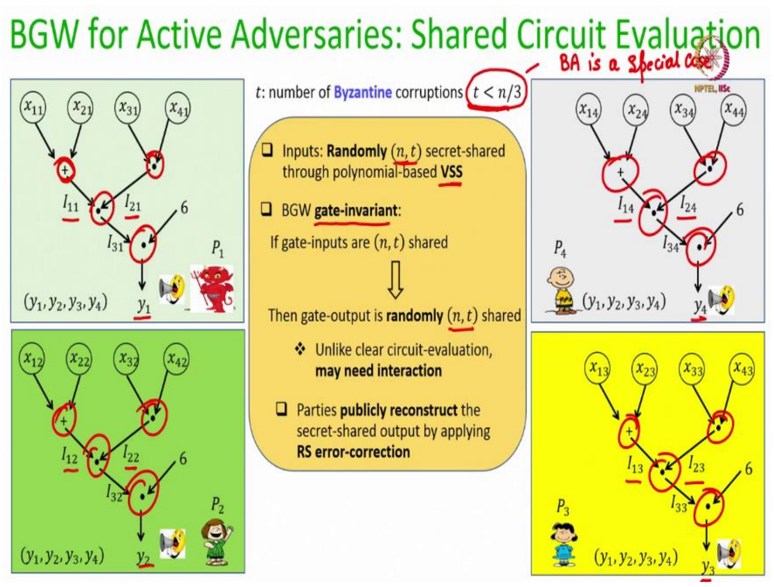
BA is a special case

t: number of Byzantine corruptions $t < n/3$

- Inputs: **Randomly** (n, t) secret-shared through polynomial-based **VSS**

The shares which it is going to distribute to the honest parties they are going to lie on some t degree polynomial and that is guaranteed because of the strong commitment property.

(Refer Slide Time: 04:01)



BGW for Active Adversaries: Shared Circuit Evaluation

BA is a Special case

t: number of Byzantine corruptions $t < n/3$

- Inputs: **Randomly** (n, t) secret-shared through polynomial-based **VSS**
- BGW **gate-invariant**:

  If gate-inputs are (n, t) shared

  $\Downarrow$

  Then gate-output is **randomly** (n, t) shared

  - Unlike clear circuit-evaluation, **may need interaction**
- Parties **publicly reconstruct** the secret-shared output by applying **RS error-correction**

NPTEL, IISc
x_{11} x_{21} x_{31} x_{41}
l_{11} l_{21} l_{31} 6
P_1
(y_1, y_2, y_3, y_4) y_1
x_{14} x_{24} x_{34} x_{44}
l_{14} l_{24} l_{34} 6
P_4
(y_1, y_2, y_3, y_4) y_4
x_{12} x_{22} x_{32} x_{42}
l_{12} l_{22} l_{32} 6
P_2
(y_1, y_2, y_3, y_4) y_2
x_{13} x_{23} x_{33} x_{43}
l_{13} l_{23} l_{33} 6
P_3
(y_1, y_2, y_3, y_4) y_3

In parallel in the same input stage party P_2 is going to act as a dealer and run or invoke an instance of the sharing phase protocol of the polynomial based VSS to secret share its input x_2 due to which every party will now have a share of the input x_2. Similarly the third party will act as a dealer and invoke the sharing phase protocol of the VSS scheme to secret share x_3 and similarly party P_4 in parallel will invoke the sharing phase protocol to secret share the input x_4 and that will complete the input stage of the protocol at the end of the which all the inputs will be (n, t) secret shared.

If the dealer is honest, then it will be also additionally guaranteed that the (n, t) secret sharing is a random secret sharing namely the probability distribution of the shares corresponding to the corrupt parties in that instance will be independent of the underlying secret which has been shared by the honest dealer.

So, for instance in this example I am considering P_1 to be corrupt. So, P_1 will be knowing all the shares of x_1 because it itself is acting as the dealer, but the share which it receives for corresponding to x_2, x_3, x_4 they will be independent of x_2, x_3, x_4. So, P_1 will completely be clueless what exactly is x_2 what exactly is x_3 and what exactly is x_4.

Now, once the input stage is over the parties go to the computation stage where they start evaluating each gate in the circuit in a topological order by maintaining the BGW gate invariant and the gate invariant is the following. If the inputs of the gate are (n, t) secret shared, then through the steps of the BGW protocol it will be guaranteed that even the gate output is randomly anti secret shared among the parties.

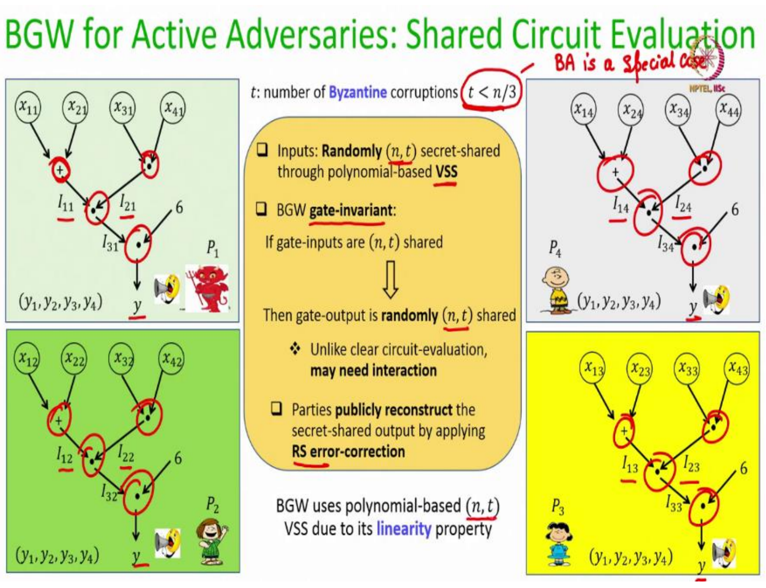
Now, maintaining this invariant may require communication interaction among the parties or it may not require interaction among the parties depending upon the type of the gate. So, again what it means here? So, in the topological order suppose this is the first gate. So, everyone will try to evaluate this first gate which is the addition gate and currently the inputs for this plus gate are x_1 and x_2.

But no one knows the value of x_1 and x_2 completely of course, the respective input owners they know the values of x_1 and x_2, but rather, but what we can say now is that the inputs x_1 and x_2 are rather currently secret shared. So, as per the steps of the BGW protocol the gate invariant will guarantee that after the evaluation of this plus gate every party obtains a share for this out for the output of this plus gate namely P_1 will have a share I_{11}, P_2 will have a share I_{12}, P_3 will have some share I_{13} and P_4 will have some share I_{14}.

So, that together I_{11}, I_{12}, I_{13} and I_{14} constitute an (n, t) secret sharing for the value I_1 then the parties go to the next gate and they evaluate it using the steps of the BGW protocol and at the end of the evaluation of this gate every party will have a share corresponding to the output of this gate.

Then they go to the next gate and then again, the invariant will ensure that every party obtains a shares corresponding to the output of this gate and then they go to the final gate and again by maintaining the invariant they obtain a share for the output value. Now, once all the gates are evaluated the parties go to the output stage where now every party can make public is its share corresponding to the output of the function there could be up to t parties who may make public incorrect shares and identity of the corrupt shares may not be known.

(Refer Slide Time: 08:18)



BGW for Active Adversaries: Shared Circuit Evaluation

BA is a Special case

t: number of Byzantine corruptions $t < n/3$

- Inputs: Randomly (n, t) secret-shared through polynomial-based VSS
- BGW gate-invariant:

  If gate-inputs are (n, t) shared

  $\Downarrow$

  Then gate-output is randomly (n, t) shared

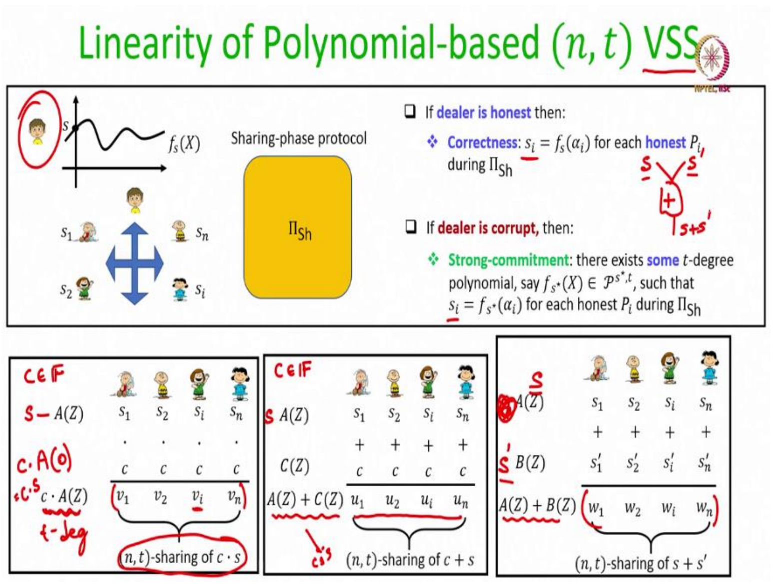
  - Unlike clear circuit-evaluation, may need interaction
- Parties publicly reconstruct the secret-shared output by applying RS error-correction

BGW uses polynomial-based (n, t) VSS due to its linearity property

x_{11} x_{21} x_{31} x_{41} I_{11} I_{21} I_{31} 6 y P_1 (y_1, y_2, y_3, y_4)
x_{12} x_{22} x_{32} x_{42} I_{12} I_{22} I_{32} 6 y P_2 (y_1, y_2, y_3, y_4)
x_{13} x_{23} x_{33} x_{43} I_{13} I_{23} I_{33} 6 y P_3 (y_1, y_2, y_3, y_4)
x_{14} x_{24} x_{34} x_{44} I_{14} I_{24} I_{34} 6 y P_4 (y_1, y_2, y_3, y_4)
NPTEL, IISc

So, what the parties can do now is that they can apply the Reed Solomon error correction and they will try to reconstruct the correct output y and terminate the protocol. So, the BGW protocol uses the polynomial based (n, t) VSS the verifiable secret sharing scheme based on polynomials because of its linearity property.

(Refer Slide Time: 08:43)



Linearity of Polynomial-based (n, t) VSS

- If dealer is honest then:
  - Correctness: $s_i = f_s(\alpha_i)$ for each honest P_i, during Π_{Sh}
- If dealer is corrupt, then:
  - Strong-commitment: there exists some t-degree polynomial, say $f_{s^\star}(X) \in \mathcal{P}^{s^\star, t}$, such that $s_i = f_{s^\star}(\alpha_i)$ for each honest P_i during Π_{Sh}

$c \in \mathbb{F}$				
s — $A(Z)$	s_1	s_2	s_i	s_n
$c \cdot A(0)$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
	c	c	c	c
$c \cdot s$ $c \cdot A(Z)$ t-deg	v_1	v_2	v_i	v_n

(n, t)-sharing of $c \cdot s$

$c \in \mathbb{F}$				
s $A(Z)$	s_1	s_2	s_i	s_n
	$+$	$+$	$+$	$+$
$C(Z)$	c	c	c	c
$A(Z) + C(Z)$ $c+s$	u_1	u_2	u_i	u_n

(n, t)-sharing of $c + s$

s $A(Z)$	s_1	s_2	s_i	s_n
	$+$	$+$	$+$	$+$
s' $B(Z)$	s'_1	s'_2	s'_i	s'_n
$A(Z) + B(Z)$	w_1	w_2	w_i	w_n

(n, t)-sharing of $s + s'$

So, what exactly is this linearity property let us quickly go through it. So, if we are using any polynomial based VSS scheme, then it guarantees the correctness and the strong commitment property the correctness guarantees that the shares of the honest parties lie on some t degree polynomial.

And a strong commitment property also guarantees that the shares of the honest parties at the end of the sharing phase also lie on a t degree polynomial; that means, if some value is secret shared using this polynomial based VSS, then irrespective of whether the dealer for that sharing instance is honest or corrupt the shares of the honest parties are guaranteed to lie on some t degree polynomial.

So, now imagine there is some value s which is (n, t) secret shared when I say (n, t) secret shared means that there exists some t degree polynomial whose constant term is that value which is shared, and every party has the value of that sharing polynomial at α_i. Namely the party P_i sorry to be more precise the party P_i has the value of that sum is sharing polynomial at α_i.

So, say there is some value s which is secret shared in this fashion and say the underlying secret sharing polynomial is A polynomial which is a t degree polynomial and suppose there is some value c in the field which is publicly known now if every party just multiplies its share of s with this value c and then if we focus on the resultant vector of shares this resultant vector of shares constitute a vector of shares lying on a t degree polynomial which is c times the A polynomial.

Why? Because if we evaluate this new polynomial at α_i we get the ith component in the new vector and what will be the constant term of this new t degree polynomial? The constant term of this new t degree polynomial will be c times the constant term of the A polynomial the constant term of the A polynomial is s that is why the constant term of this new t degree polynomial is $c \cdot s$; that means, the new vector of shares constitutes a vector of (n, t) secret sharing for the value $c \cdot s$.

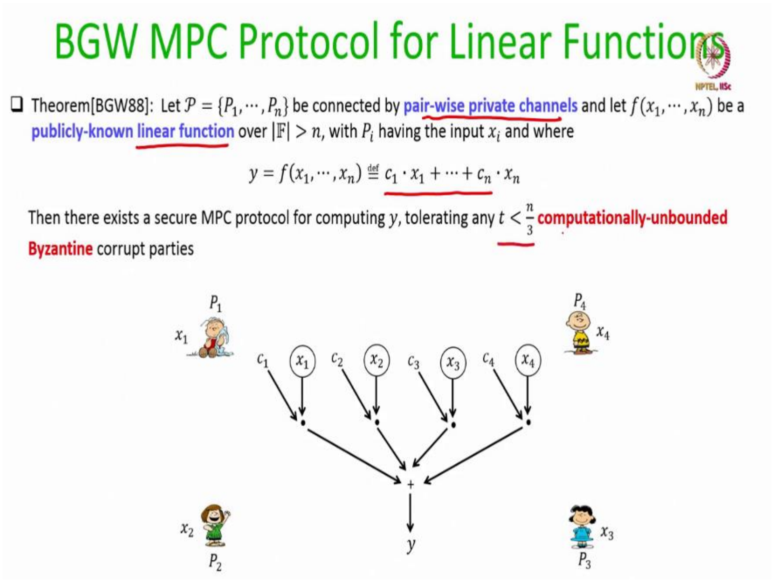
In the same spirit if we have two sorry if we have a value s which has been secret shared using this polynomial based VSS and if there is some value c from the field and now every party adds its respective share of s with the value c. Then we obtain a vector of values where the party P_i will have the ith component of the vector and together this vector now constitutes an (n, t) secret sharing for the value $c + s$.

Because this new vector of values lie on a new t degree polynomial namely this polynomial A plus a constant polynomial C and the constant term of this new t degree polynomial is $c + s$. And finally, if there are two values s and s' and if every party locally adds its respective share of s and s' then that will give that party is share of $s + s'$ because together this new vector of values lie on a new t degree polynomial whose constant term is $s + s'$ and ith party will have the ith component of this new vector.

So, this is what is linearity property what this shows here is that if you have a value or more than one value which is secret shared using A polynomial based VSS in an (n, t) secret shared fashion. And if you want to compute some linear function of those secret shared values then you can compute that non interactively.

Namely, say for instance if s is secret shared and s' is secret shared and if there is an addition gate in the circuit whose inputs would have been s and s' then during the VGW gate invariant the parties need not have to interact each party can locally go and add its share of s with its share of s' that will give that party its share of s + s' and so on; that is, what is the linearity property here.

(Refer Slide Time: 13:45)



BGW MPC Protocol for Linear Functions

- Theorem[BGW88]: Let $\mathcal{P} = \{P_1, \cdots, P_n\}$ be connected by pair-wise private channels and let $f(x_1, \cdots, x_n)$ be a publicly-known linear function over $|\mathbb{F}| > n$, with P_i having the input x_i and where

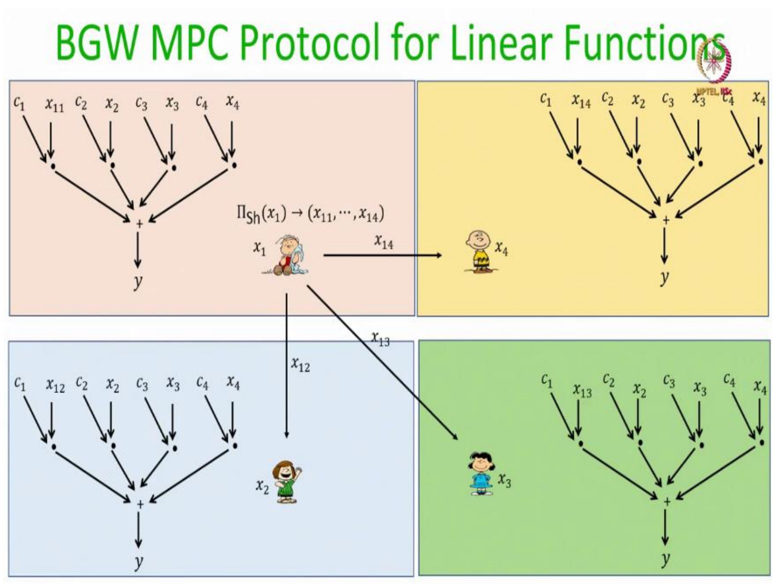
$$y = f(x_1, \cdots, x_n) \overset{\text{def}}{=} c_1 \cdot x_1 + \cdots + c_n \cdot x_n$$

Then there exists a secure MPC protocol for computing y, tolerating any $t < \frac{n}{3}$ computationally-unbounded Byzantine corrupt parties

Now, by exploiting this linearity property we can quickly get the following result. Imagine you have a set of parties connected by pair wise private channels and imagine you have a function which is represented by and imagine you have a function which is now a linear function over the field namely the function is of this form where $c_1, c_2, \dots, c_n$ are publicly known constants and this function is represented by the following circuit.

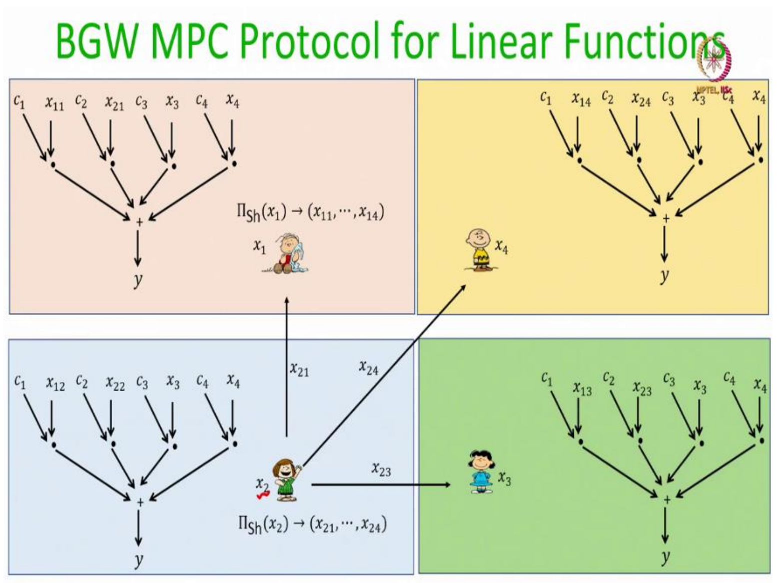
Then there exists a perfectly secure MPC protocol, which allows the parties to securely compute this function even if there are up to $\frac{n}{3}$ computationally unbounded maliciously corrupt parties.

(Refer Slide Time: 14:31)


BGW MPC Protocol for Linear Functions
$\Pi_{Sh}(x_1) \rightarrow (x_{11}, \cdots, x_{14})$
x_1
x_{14}
x_{13}
x_{12}
x_2
x_3
x_4
c_1 x_{11} c_2 x_2 c_3 x_3 c_4 x_4
c_1 x_{14} c_2 x_2 c_3 x_3 c_4 x_4
c_1 x_{12} c_2 x_2 c_3 x_3 c_4 x_4
c_1 x_{13} c_2 x_2 c_3 x_3 c_4 x_4
$+$
y

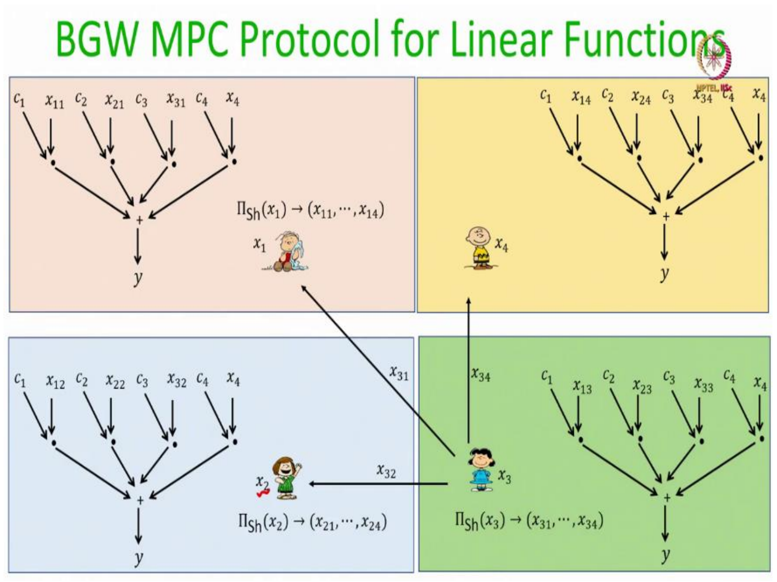
And the protocol is very straight forward during the input stage every party will invoke an instance of the secret sharing phase protocol. The sharing phase protocol of a verifiable secret sharing scheme based on polynomials. So, for instance, P_1 will act as a dealer it will run the sharing phase protocol of the VSS and say the resultant shares are $x_{11}, x_{12}, \dots, x_{1n}$, which are distributed to the respected parties, during the sharing phase protocol.

(Refer Slide Time: 15:13)


BGW MPC Protocol for Linear Functions
c_1 x_{11} c_2 x_{21} c_3 x_3 c_4 x_4
$\Pi_{Sh}(x_1) \rightarrow (x_{11}, \cdots, x_{14})$
x_1
y
c_1 x_{14} c_2 x_{24} c_3 x_3 c_4 x_4
x_4
c_1 x_{12} c_2 x_{22} c_3 x_3 c_4 x_4
x_{21}
x_{24}
x_{23}
x_2
$\Pi_{Sh}(x_2) \rightarrow (x_{21}, \cdots, x_{24})$
c_1 x_{13} c_2 x_{23} c_3 x_3 c_4 x_4
x_3

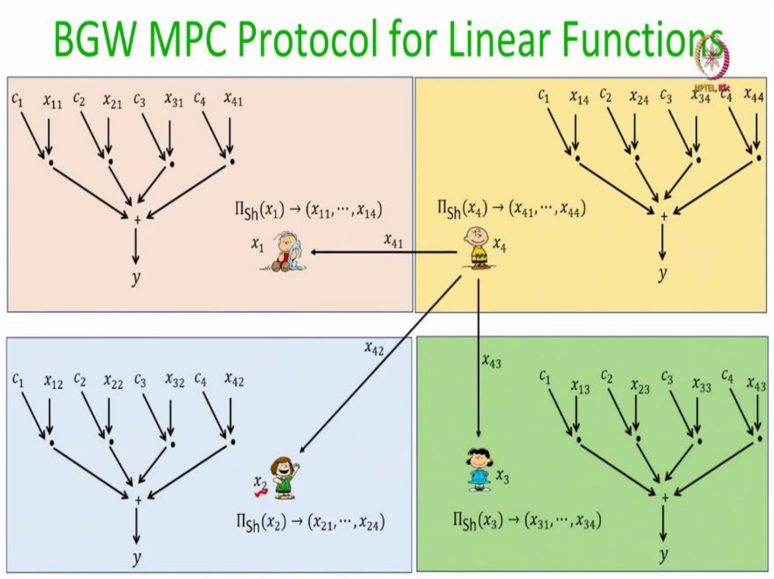
In the same spirit party P_2 acts as a dealer and this can happen in parallel. So, we do not need to depend on or we do not need to wait for the VSS instance of party P_1 to get over and then start the VSS instance for P_2 though all these VSS instances can be executed in parallel. So, in parallel P_2 can act as a 0 with input x_2 and trigger or invoke an instance of the sharing phase protocol of the polynomial based VSS and that will ensure that its input x_2 is (n, t) secret shared.

(Refer Slide Time: 15:48)

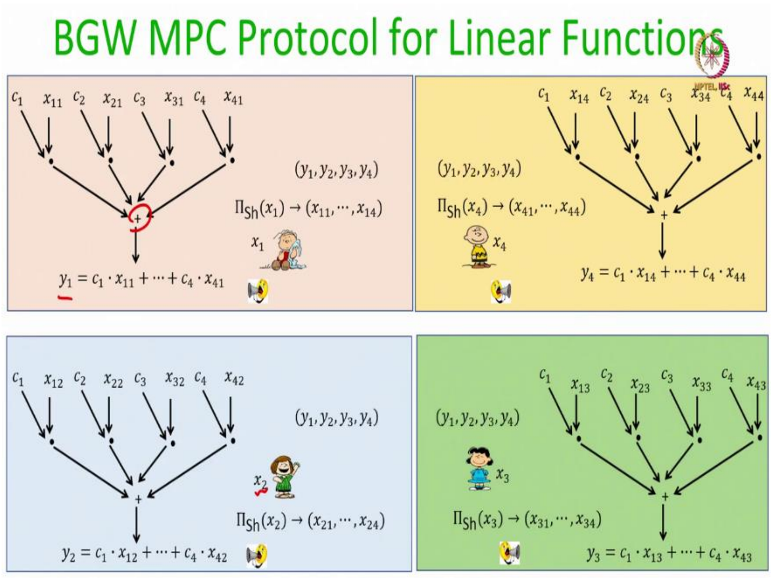

BGW MPC Protocol for Linear Functions
c_1 x_{11} c_2 x_{21} c_3 x_{31} c_4 x_4
$\Pi_{Sh}(x_1) \rightarrow (x_{11}, \cdots, x_{14})$
x_1
y
c_1 x_{14} c_2 x_{24} c_3 x_{34} c_4 x_4
x_4
c_1 x_{12} c_2 x_{22} c_3 x_{32} c_4 x_4
x_{31}
x_{34}
x_{32}
x_2
$\Pi_{Sh}(x_2) \rightarrow (x_{21}, \cdots, x_{24})$
c_1 x_{13} c_2 x_{23} c_3 x_{33} c_4 x_4
x_3
$\Pi_{Sh}(x_3) \rightarrow (x_{31}, \cdots, x_{34})$

In parallel P_3 act as a dealer and do the same does the same and P_4 acts as a dealer and has the same.

(Refer Slide Time: 15:54)

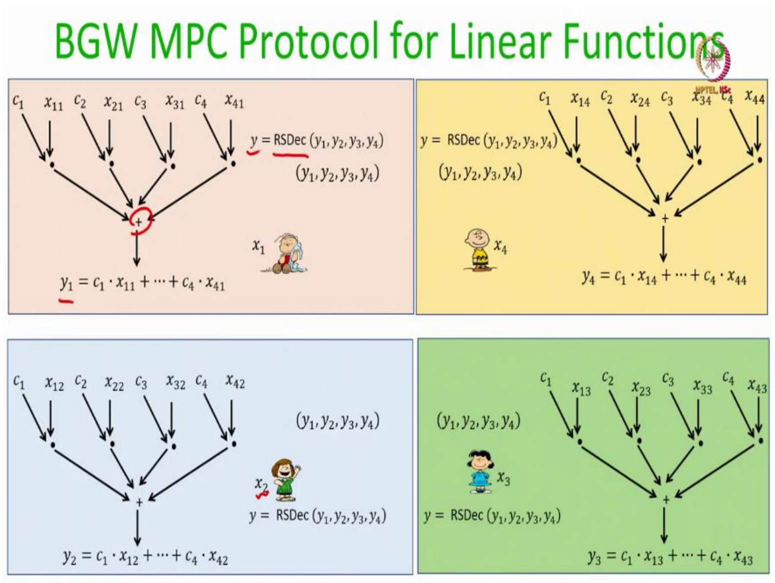

BGW MPC Protocol for Linear Functions
c_1 x_{11} c_2 x_{21} c_3 x_{31} c_4 x_{41}
$\Pi_{Sh}(x_1) \rightarrow (x_{11}, \cdots, x_{14})$
x_1
x_{41}
y
c_1 x_{14} c_2 x_{24} c_3 x_{34} c_4 x_{44}
$\Pi_{Sh}(x_4) \rightarrow (x_{41}, \cdots, x_{44})$
x_4
y
x_{42}
c_1 x_{12} c_2 x_{22} c_3 x_{32} c_4 x_{42}
x_2
$\Pi_{Sh}(x_2) \rightarrow (x_{21}, \cdots, x_{24})$
y
x_{43}
c_1 x_{13} c_2 x_{23} c_3 x_{33} c_4 x_{43}
x_3
$\Pi_{Sh}(x_3) \rightarrow (x_{31}, \cdots, x_{34})$
y

(Refer Slide Time: 16:02)


BGW MPC Protocol for Linear Functions
c_1 x_{11} c_2 x_{21} c_3 x_{31} c_4 x_{41}
(y_1, y_2, y_3, y_4)
$\Pi_{Sh}(x_1) \rightarrow (x_{11}, \cdots, x_{14})$
x_1
$y_1 = c_1 \cdot x_{11} + \cdots + c_4 \cdot x_{41}$
c_1 x_{14} c_2 x_{24} c_3 x_{34} c_4 x_{44}
(y_1, y_2, y_3, y_4)
$\Pi_{Sh}(x_4) \rightarrow (x_{41}, \cdots, x_{44})$
x_4
$y_4 = c_1 \cdot x_{14} + \cdots + c_4 \cdot x_{44}$
c_1 x_{12} c_2 x_{22} c_3 x_{32} c_4 x_{42}
(y_1, y_2, y_3, y_4)
x_2
$\Pi_{Sh}(x_2) \rightarrow (x_{21}, \cdots, x_{24})$
$y_2 = c_1 \cdot x_{12} + \cdots + c_4 \cdot x_{42}$
c_1 x_{13} c_2 x_{23} c_3 x_{33} c_4 x_{43}
(y_1, y_2, y_3, y_4)
x_3
$\Pi_{Sh}(x_3) \rightarrow (x_{31}, \cdots, x_{34})$
$y_3 = c_1 \cdot x_{13} + \cdots + c_4 \cdot x_{43}$

Now, once all the inputs are secret shared, we have only one linear gate in the circuit. So, parties can use the linearity property and obtain their respective shares for this for the output of this linear gate and then they can make public their respective shares of the output of the circuit. The corrupt parties may make public incorrect shares.

(Refer Slide Time: 16:21)

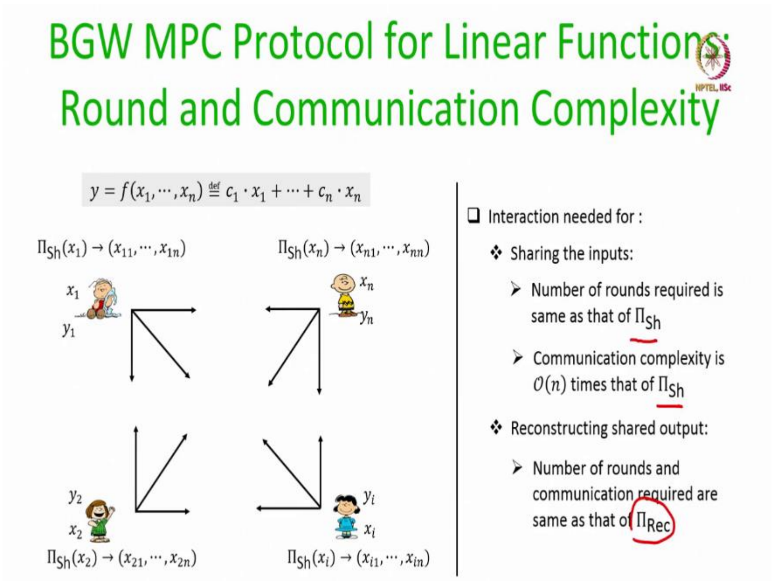


BGW MPC Protocol for Linear Functions

c_1 x_{11} c_2 x_{21} c_3 x_{31} c_4 x_{41}
y = RSDec (y_1, y_2, y_3, y_4)
(y_1, y_2, y_3, y_4)
x_1
$y_1 = c_1 \cdot x_{11} + \cdots + c_4 \cdot x_{41}$
c_1 x_{14} c_2 x_{24} c_3 x_{34} c_4 x_{44}
y = RSDec (y_1, y_2, y_3, y_4)
(y_1, y_2, y_3, y_4)
x_4
$y_4 = c_1 \cdot x_{14} + \cdots + c_4 \cdot x_{44}$
c_1 x_{12} c_2 x_{22} c_3 x_{32} c_4 x_{42}
(y_1, y_2, y_3, y_4)
x_2
y = RSDec (y_1, y_2, y_3, y_4)
$y_2 = c_1 \cdot x_{12} + \cdots + c_4 \cdot x_{42}$
c_1 x_{13} c_2 x_{23} c_3 x_{33} c_4 x_{43}
(y_1, y_2, y_3, y_4)
x_3
y = RSDec (y_1, y_2, y_3, y_4)
$y_3 = c_1 \cdot x_{13} + \cdots + c_4 \cdot x_{43}$

So, what the parties can do is they can apply the Reed Solomon decoding algorithm on the vector of shares corresponding to the output gate and reconstruct back the function output. That is the simple BGW protocol for linear functions.

(Refer Slide Time: 16:33)



BGW MPC Protocol for Linear Functions: Round and Communication Complexity

$y = f(x_1, \cdots, x_n) \overset{\text{def}}{=} c_1 \cdot x_1 + \cdots + c_n \cdot x_n$

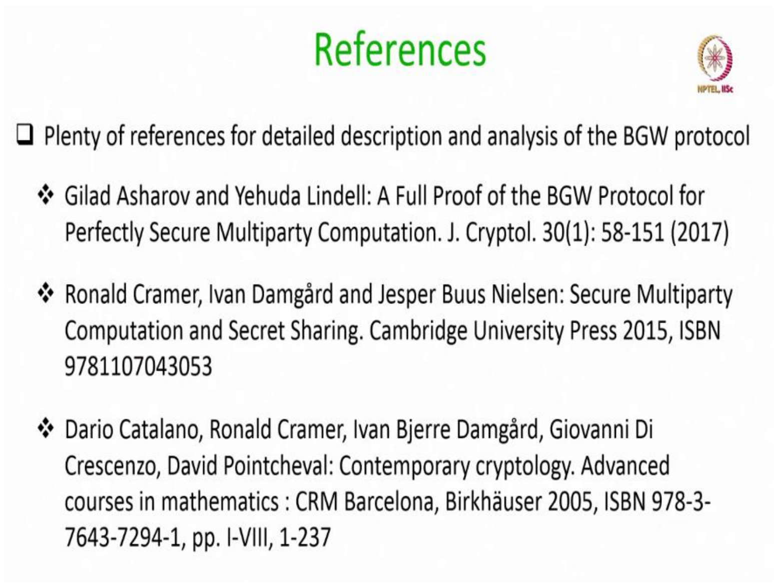
$\Pi_{\mathsf{Sh}}(x_1) \rightarrow (x_{11}, \cdots, x_{1n})$
$\Pi_{\mathsf{Sh}}(x_n) \rightarrow (x_{n1}, \cdots, x_{nn})$
x_1
y_1
x_n
y_n
y_2
x_2
y_i
x_i
$\Pi_{\mathsf{Sh}}(x_2) \rightarrow (x_{21}, \cdots, x_{2n})$
$\Pi_{\mathsf{Sh}}(x_i) \rightarrow (x_{i1}, \cdots, x_{in})$

- Interaction needed for :
  - Sharing the inputs:
    - Number of rounds required is same as that of Π_{Sh}
    - Communication complexity is $\mathcal{O}(n)$ times that of Π_{Sh}
  - Reconstructing shared output:
    - Number of rounds and communication required are same as that of Π_{Rec}

Let us do the quick analysis of the round and communication complexity. So, interaction is needed in this protocol during the input stage for verifiably secret sharing the inputs. So, the number of rounds in the input stage is basically the same as the number of rounds in the sharing phase protocol of the underlying VSS scheme and the communication required will be n times the communication required for one instance of the sharing phase protocol.

During the computation stage no interaction is involved and again interaction is involved during the reconstruction stage when the parties exchange their shares of the function output and that will require the number of rounds, which will be same as the number of rounds for the reconstruction function of the underlying secret sharing scheme.

(Refer Slide Time: 17:32)



References

- ❑ Plenty of references for detailed description and analysis of the BGW protocol
  - ❖ Gilad Asharov and Yehuda Lindell: A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation. J. Cryptol. 30(1): 58-151 (2017)
  - ❖ Ronald Cramer, Ivan Damgård and Jesper Buus Nielsen: Secure Multiparty Computation and Secret Sharing. Cambridge University Press 2015, ISBN 9781107043053
  - ❖ Dario Catalano, Ronald Cramer, Ivan Bjerre Damgård, Giovanni Di Crescenzo, David Pointcheval: Contemporary cryptology. Advanced courses in mathematics : CRM Barcelona, Birkhäuser 2005, ISBN 978-3-7643-7294-1, pp. I-VIII, 1-237

So, that is a very simple MPC protocol perfectly secure MPC protocol for linear functions in the next lecture we will do an analysis security analysis of the protocol. There are plenty of references for detailed description and a very rigorous form on analysis of the perfectly secure BGW protocol my personal favorite is this first reference.

Thank you.