

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

Module - 4
Lecture - 20
The BGW MPC Protocol for Linear Functions

(Refer Slide Time: 00:33)

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 start going into the details of the BGW protocol, the exact details. And we will start with a simple case, namely, we will assume a function which just consists of linear gates.

(Refer Slide Time: 00:50)

BGW Approach: Shared Circuit Evaluation

P_1

P_4

P_2

P_3

t : number of (semi-honest) corruptions

- Inputs: **Randomly** (n, t) secret-shared
- BGW gate-invariant: ?
If gate-inputs are randomly (n, t) shared
Then gate-output is **randomly** (n, t) shared
- ❖ Unlike clear circuit-evaluation, may need interaction
- Parties **publicly reconstruct** the secret-shared output
- BGW uses (n, t) Shamir SS
- ❖ Gate-invariant can be maintained freely (no interaction)

Just to recap; this was the shared circuit-evaluation approach proposed by BGW, and all the generic MPC protocols follow this blueprint of shared circuit-evaluation. Namely, it ensures that the inputs are secret-shared and all the intermediate values in the computation are also secret-shared. And then, finally, you go and publicly reconstruct the function output. What will be different in different MPC protocols?

What exactly is the secret-sharing scheme you are following? How exactly the inputs are shared? And how exactly the gate-invariant is maintained? But the approach or the philosophy remains the same in all the generic MPC protocol. So, the BGW MPC protocols follows the (n, t) Shamir secret-sharing for instantiating this (n, t) secret-sharing in this blueprint. And the reason it uses (n, t) Shamir secret-sharing is that, it ensures that the gate-invariant can be maintained without requiring any interaction among the parties.

So, remember, I said that maintaining this invariant may require interaction among the parties, depending upon the type of the gate; because, in the arithmetic circuit, we can have various types of gates. What I am saying here is that, if we use Shamir secret-sharing to instantiate this (n, t) secret-sharing, then, maintaining this invariant does not require any interaction among the parties, if the gate which needs to be evaluated is a linear gate. And this comes from the linearity property of your Shamir secret-sharing.


(Refer Slide Time: 02:42)

Linearity of Shamir's (n, t) Sharing (evaluation points)

- Public set-up: finite field $(\mathbb{F}, +, \cdot)$, with $|\mathbb{F}| > n$ and publicly known, non-zero distinct elements $\alpha_1, \dots, \alpha_n \in \mathbb{F}$

$\text{Sh}_{\text{Shamir}}(s)$ | \mathbb{F} | = n

- Randomly pick $a_1, \dots, a_t \in \mathbb{F}$
- Define $A(Z) \triangleq s + a_1 \cdot Z + \dots + a_t \cdot Z^t$
($A(Z) \in \mathbb{F}^{(p, s, t)}$ set of all poly of deg t with s as the constant terms)
- For $i = 1, \dots, n$, share $s_i \triangleq A(\alpha_i)$



So, let us quickly see, recap the linearity property of (n, t) Shamir secret-sharing. This was your Shamir secret-sharing algorithm. If s is the value which needs to be secret-shared, where s is an element from the field, then, the public setup is a finite field whose cardinality is more

than n and the public knowledge of n distinct non-zero evaluation points, $\alpha_1, \dots, \alpha_n$. And these evaluation points will be fixed once for all, for all the instantiations of Shamir secret-sharing, for computing the shares.

So, now, if s is the value which needs to be Shamir secret-shared, what we do is, we randomly pick a t -degree polynomial $y = s + a_1x + a_2x^2 + \dots + a_tx^t$ whose constant term is the value s . That means, the constant term s is fixed, but all other remaining coefficients a_1, a_2, \dots, a_t , they are randomly picked from the field. And when I say a polynomial of degree- t , that does not mean that a_t is not allowed to be 0.

I am following the convention that, when I say a degree- t , that means, there are $t + 1$ coefficients altogether. The constant term will be the secret, the remaining coefficients can be 0, non-zero; they are any elements from the field. So, even if they are all zeros, altogether, I will consider it as a vector of $t + 1$ coefficients; and hence, a polynomial of degree- t . That is the nomenclature I am following here.

Also remember that this set $\mathcal{P}^{(s,t)}$, it denotes the set of all polynomials of degree- t with s as the constant term. All these things, we have discussed rigorously in our earlier lectures. And we know that the number of such polynomials is nothing but the cardinality of field raised to the power t . So, to share the value s , one such polynomial is picked uniformly at random, which is equivalent to saying that, pick your coefficients a_1 to a_t uniformly at random.

And the shares are computed by evaluating this polynomial at these evaluation points $\alpha_1, \dots, \alpha_n$. And the i^{th} share will be the evaluation of the polynomial at α_i . I stress, every party will know that the i^{th} share is computed by evaluating this random polynomial at α_i . That is not private information, because the algorithm is publicly-known.

(Refer Slide Time: 05:37)

Linearity of Shamir's (n, t) Sharing (evaluation points)

Public set-up: finite field $(\mathbb{F}, +, \cdot)$, with $|\mathbb{F}| > n$ and publicly known, non-zero distinct elements $\alpha_1, \dots, \alpha_n \in \mathbb{F}$

ShShamir(s)

- ☐ Randomly pick $a_1, \dots, a_t \in \mathbb{F}$
- ☐ Define $A(Z) \stackrel{\text{def}}{=} s + a_1 \cdot Z + \dots + a_t \cdot Z^t$
 $A(Z) \in_{\mathcal{P}^{s,t}}$
- ☐ For $i = 1, \dots, n$, share $s_i \stackrel{\text{def}}{=} A(\alpha_i)$

$A(Z)$	s_1	s_2	s_i	s_n

	c	c	c	c
$c \cdot A(Z)$	v_1	v_2	v_i	v_n
	} (n, t) -sharing of $c \cdot s$			

→ public constants



It is only the polynomial which is randomly decided by the dealer at the time of secret-sharing the value s . So, the linearity property means here the following: If you have a value s which has been secret-shared through some random a -degree polynomial A , and if there is some public constant c , then, each party, if it multiplies its respective share of s with the same value c ; each party does it locally; then, all together, they will obtain a vector of shares.

Together, they obtain a vector of shares, I mean, each party will have the i^{th} component of that resultant vector; but as a whole, I am considering it as a vector. That resultant vector will be now (n, t) secret-sharing of the value $c \cdot s$. That means, without even knowing the value s , just with the knowledge of c , each party can compute its share of $c \cdot s$. What does it have to do? Just go and multiply its share of s with the constant c ; that is all.

(Refer Slide Time: 06:47)

Linearity of Shamir's (n, t) Sharing (evaluation points)

Public set-up: finite field $(\mathbb{F}, +, \cdot)$, with $|\mathbb{F}| > n$ and publicly known, non-zero distinct elements $\alpha_1, \dots, \alpha_n \in \mathbb{F}$

ShShamir(s)

- ☐ Randomly pick $a_1, \dots, a_t \in \mathbb{F}$
- ☐ Define $A(Z) \stackrel{\text{def}}{=} s + a_1 \cdot Z + \dots + a_t \cdot Z^t$
 $A(Z) \in_{\mathcal{P}^{s,t}}$
- ☐ For $i = 1, \dots, n$, share $s_i \stackrel{\text{def}}{=} A(\alpha_i)$

ShShamir(s')

- ☐ Randomly pick $b_1, \dots, b_t \in \mathbb{F}$
- ☐ Define $B(Z) \stackrel{\text{def}}{=} s' + b_1 \cdot Z + \dots + b_t \cdot Z^t$
 $B(Z) \in_{\mathcal{P}^{s',t}}$
- ☐ For $i = 1, \dots, n$, share $s'_i \stackrel{\text{def}}{=} B(\alpha_i)$

$A(Z)$	s_1	s_2	s_i	s_n

	c	c	c	c
$c \cdot A(Z)$	v_1	v_2	v_i	v_n
	} (n, t) -sharing of $c \cdot s$			

$A(Z)$	s_1	s_2	s_i	s_n
	+	+	+	+
$C(Z)$	c	c	c	c
$A(Z) + C(Z)$	u_1	u_2	u_i	u_n
	} (n, t) -sharing of $c + s$			

$A(Z)$	s_1	s_2	s_i	s_n
	+	+	+	+
$B(Z)$	s'_1	s'_2	s'_i	s'_n
$A(Z) + B(Z)$	w_1	w_2	w_i	w_n
	} (n, t) -sharing of $s + s'$			

In the same way, if every party just adds the public constant c to its respective share of s , then, each party gets access to its share of $c + s$. And in the same way, if there is another value s' which has been secret-shared by running Shamir secret-sharing, where the sharing polynomial is say $B(Z)$, which has been picked from the set of all possible polynomials of degree- t with s' as the constant term; and now, if s_i and s'_i are the shares of s and s' respectively, for the party P_i , and if every party just goes and adds its respective share of s and s' , it will obtain its share of $s + s'$.

So, linearity here means that you can perform linear operations on the secret by performing the similar operation on the shares itself. That means, if you want to compute the shares of $c \cdot s$, multiply c with the shares of s . If you want to compute shares of $c + s$, add the value c to the shares of s . If you want to compute the shares of $s + s'$, add the shares of s and s' .

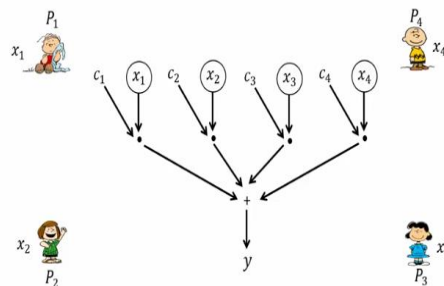
(Refer Slide Time: 08:11)

BGW MPC Protocol for Linear Functions

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

$$y = f(x_1, \dots, x_n) \triangleq c_1 \cdot x_1 + \dots + c_n \cdot x_n$$

*c_1, c_2, \dots, c_n : publicly known constants from F
 $n=4$*



So, now, with this observation, we will go and see the BGW MPC protocol for the simple case where the function that the parties want to compute is a linear function. And again, there are plenty of real-world examples, real-world computations which can be abstracted by linear functions itself. So, the setting here is the following: So, this is the theorem statement that I am quoting from the BGW paper. You have the set of parties, which I denote by \mathcal{P} .

And we are in the private channel model. And assume that there is a publicly-known function f over the field, which is a linear function. That means, it takes the inputs $x_1, x_2, \dots, x_i, \dots, x_n$ from the respective parties; and the output is $c_1 \cdot x_1 + c_2 \cdot x_2 + \dots + c_n \cdot x_n$, where

c_1, c_2, \dots, c_n are publicly-known constants from \mathbb{F} ; because, this is the form of any linear function over the field.

So, for instance, if I take $n = 4$, then the inputs are x_1, x_2, x_3 and x_4 , and this is the corresponding arithmetic circuit. So, as I said, there can be several real-world functions which can fall under these categories. If you remember the toy summation protocol; the summation protocol is basically a linear function, because your constant c_1, c_2, \dots, c_n are 1, all. So, like that, there are several interesting linear functions which the parties may want to securely compute.

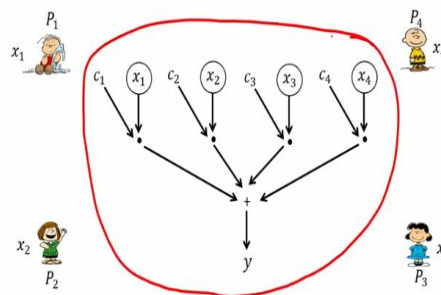
(Refer Slide Time: 10:05)

BGW MPC Protocol for Linear Functions

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

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

Then there exists a secure MPC protocol for computing y , tolerating any $t < n$ computationally-unbounded semi-honest corrupt parties

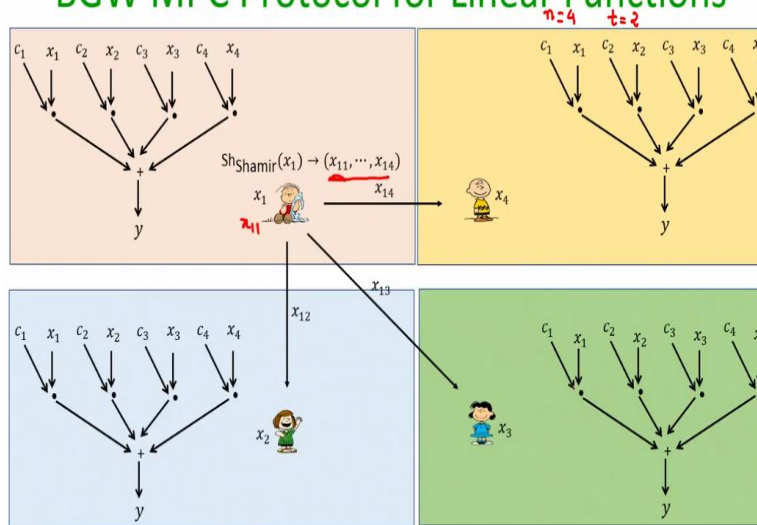


Now, what the BGW theorem says here for the linear functions is the following: If you are in the private channel model and if there is a semi-honest adversary controlling any set of t parties, where t of course has to be strictly less than n ; then, still there exists a secure MPC protocol for computing this function y . And this holds even if the t corrupt parties are computationally-unbounded. So, we are now going to see the proof of this theorem.

The proof of this theorem will be through a protocol. And then, we will analyse the security of this protocol. And now, if you are wondering, why $t < n$? Because, if I say $t = n$, then basically, if all the n parties are corrupt, then, all the inputs are revealed. So, that is why, the trivial bound for t is $t < n$. And again, what we are going to do is, we will see the BGW shared circuit-evaluation for securely evaluating this function.

(Refer Slide Time: 11:13)

BGW MPC Protocol for Linear Functions



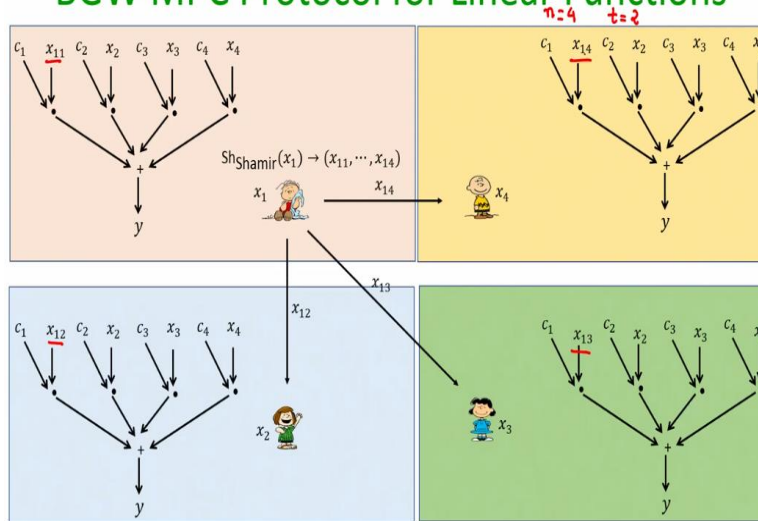
For demonstration purpose, I stick to the case where $n = 4$, but whatever I am discussing here, it generalises for any $n > t$. So, this is the circuit which parties want to securely compute. What the first party does is the following: So, remember the first stage in the shared circuit-evaluation is approaches; the inputs, respective inputs of the parties for the function needs to be secret-shared. And who should secret-share those inputs?

The input owners themselves. So, P_1 is the owner of the value x_1 . It will run an instance of (n, t) Shamir secret-sharing, where the parameter t is also publicly-known. Let us make $t = 2$, again for the sake of demonstration. So, it will pick, namely, a t -degree polynomial whose constant term is x_1 and compute the vector of these shares x_{11}, \dots, x_{1n} . In this case, n is 4; so, it will compute 4 shares.

And it will distribute the respective shares to the respective parties. So, it keeps x_{11} itself. It gives x_{12} over the secure channel to P_2 . It gives x_{13} over the secure channel to P_3 . And it gives x_{14} over the secure channel to P_4 . So, that means, now, each party, what information regarding x_1 do they have respectively?

(Refer Slide Time: 12:43)

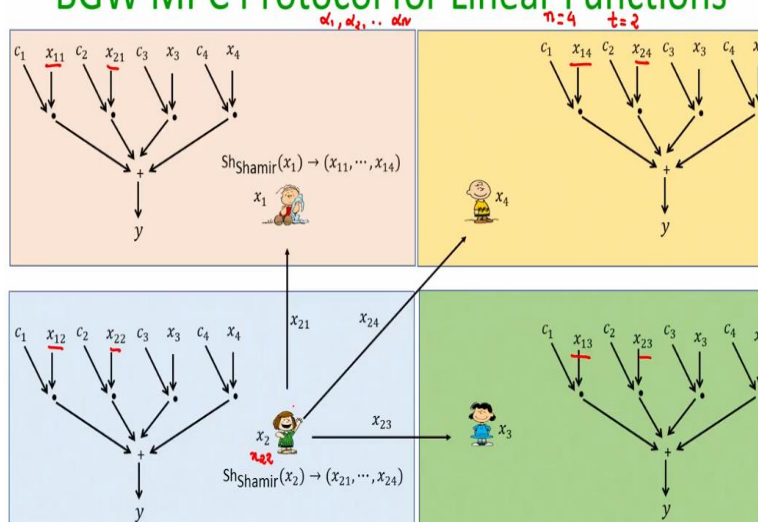
BGW MPC Protocol for Linear Functions



Each party now has its respective shares of x_1 in their local copy of the circuit.

(Refer Slide Time: 12:54)

BGW MPC Protocol for Linear Functions

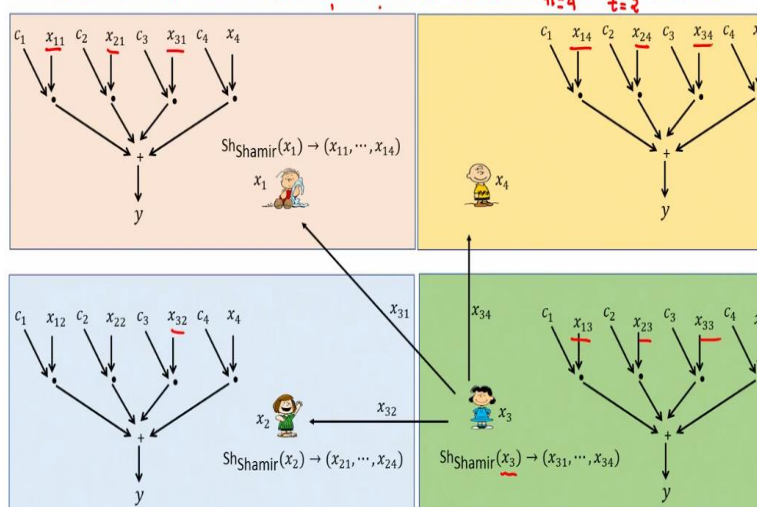


In parallel, party P_2 will act as a dealer and secret-share its input x_2 with the threshold $t = 2$. So, it will compute a vector of 4 shares; it will keep the share x_{22} with itself; and the first share of $x_2 = x_{21}$, it will give over the secure channel to P_1 and so on. And now, each party, they have got 1 share of x_2 . And remember, all the evaluations are performed over the same $\alpha_1, \dots, \alpha_n$.

That means, the $\alpha_1, \dots, \alpha_n$ which are used by P_1 to compute the shares, will be the same as the evaluation points used by P_2 to compute its shares of x_2 . P_3 also will use the same set of $\alpha_1, \dots, \alpha_n$, so on. So, $\alpha_1, \dots, \alpha_n$, they are not going to change; and that is why I am not bringing them into picture.

(Refer Slide Time: 13:53)

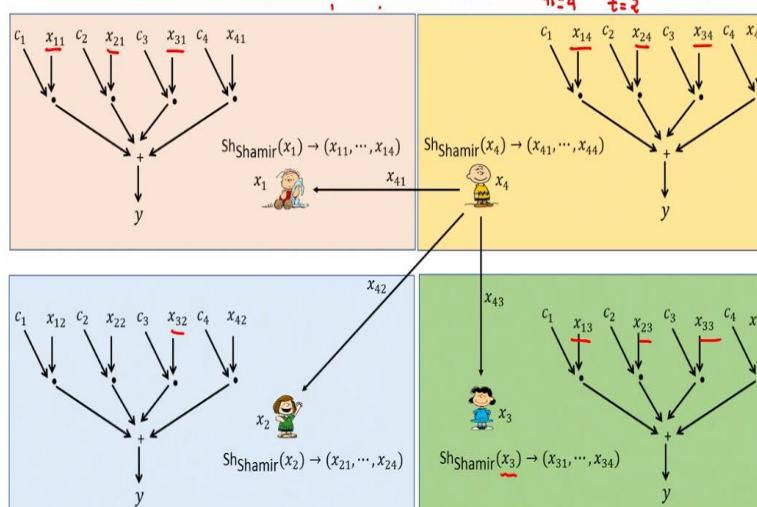
BGW MPC Protocol for Linear Functions



In parallel, P_3 will independently secret-share its input as per Shamir secret-sharing. And now, everyone will have their respective shares of x_3 .

(Refer Slide Time: 14:09)

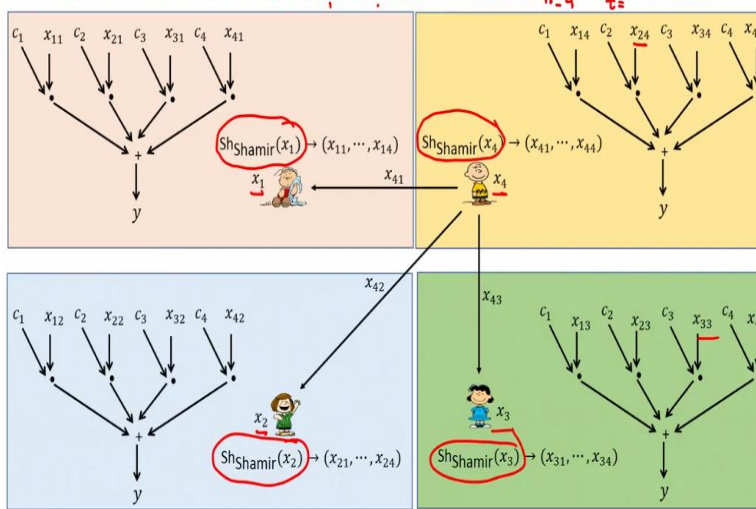
BGW MPC Protocol for Linear Functions



And P_4 will independently generate Shamir shares for the input x_4 and distribute among the parties. Now, before I proceed, let me stress here that; remember, Shamir secret-sharing is a randomised algorithm; that means, even if the parties P_1, P_2, P_3, P_4 executes this BGW MPC protocol with the same value of x_1, x_2, x_3 and x_4 , the shares which are going to be produced in every instance will be different, because they depend upon the internal randomness, namely, the sharing polynomials which are used to compute the shares.

(Refer Slide Time: 14:49)

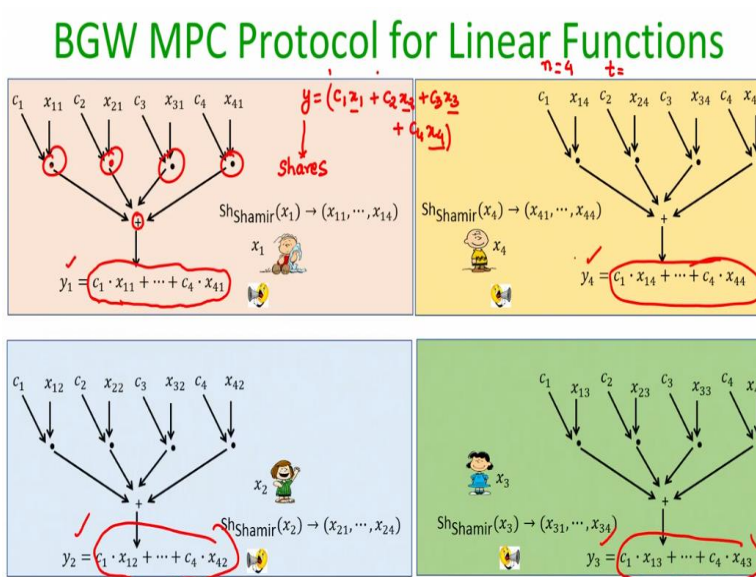
BGW MPC Protocol for Linear Functions



And it will not be the case that for sharing the same x_i , every time the same sharing polynomial will be picked. The coefficients of the sharing polynomial, they are picked uniformly at random. So, that is why, since the sharing polynomial could be picked uniformly at random, the shares themselves are going to take different values depending upon what precisely are those random coefficients.

So, this completes the input stage. All the inputs are now available in an (n, t) secret-shared fashion. Now, the parties will proceed to evaluate the gates. And what are the gates in this circuit here?

(Refer Slide Time: 15:37)



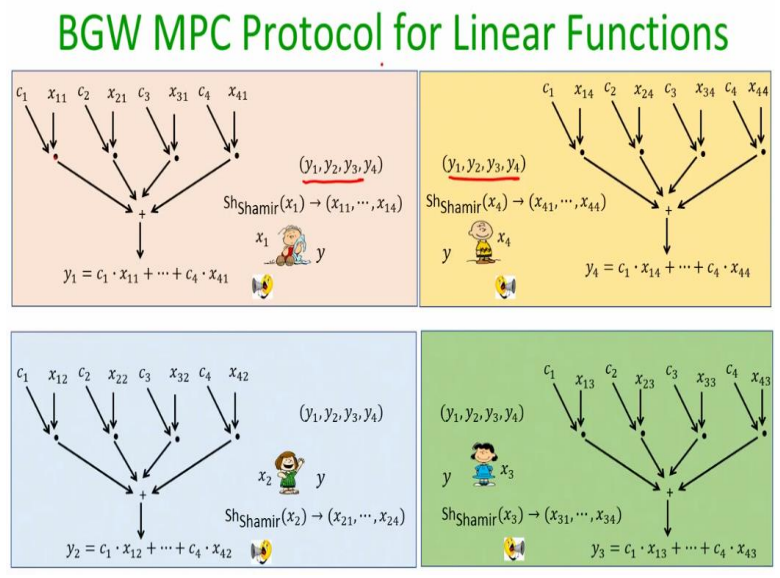
If you see closely here, all these gates are linear gates. So, this gate is a multiplication gate, but multiplication with a public constant c_1 . This second gate is also a multiplication gate, but it is

a linear gate, because c_2 is publicly-known constant and so on. And after that, there is a plus gate, which anyhow is a linear gate. So, remember, your computations which parties want to compute is, $y = c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 \cdot x_3 + c_4 \cdot x_4$.

This is the computation which parties want to securely compute. x_1, x_2, x_3, x_4 , no one knows their values right now; they are collectively secret-shared. And since this is a linear function of x_1, x_2, x_3, x_4 , by the linearity property of Shamir secret-sharing, if the same linear computation is performed on the shares of x_1, x_2, x_3, x_4 , it will result in shares of y . And that is what the parties will do.

Each party will locally compute the same linear function of the shares of x_1, x_2, x_3, x_4 . And this, they are doing locally. By saying locally, I mean, they are not interacting. Whatever values they have at their own disposal, they are computing this linear function respectively. And now, after this, there is no other gate in the circuit. That means, the computation has been performed. Now, it is the time to announce the results. So, everyone announces the result publicly. Namely, their respective shares of y_1, y_2, y_3 and y_4 , they publicly announce.

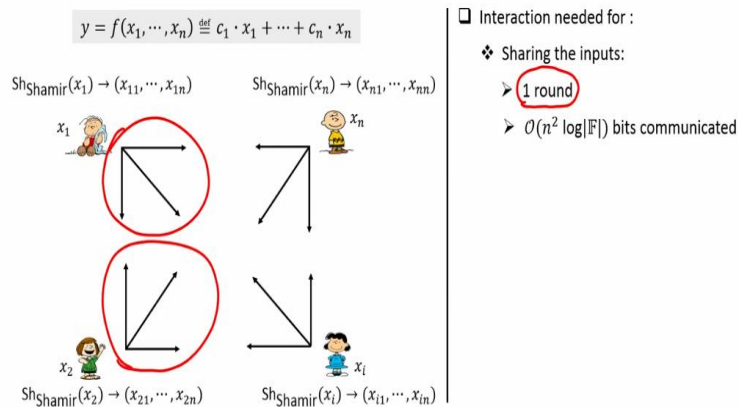
(Refer Slide Time: 17:29)



And if they now publicly announce, everyone will now have the full vector of y_1, y_2, y_3, y_4 , namely, all the shares of y . And now, they can apply the Lagrange interpolation on this vector of y shares and get back the value y . That is a BGW MPC protocol. So, now we have to analyse whether this protocol is secure or not and so on.

(Refer Slide Time: 17:59)

BGW MPC Protocol for Linear Functions: Round and Communication Complexity



But before going into that, we will first analyse the number of rounds required in the protocol and how much communication is done in the protocol. So, round complexity means the total number of rounds required in the MPC protocol. Communication complexity means how many bits are communicated in the protocol overall. And remember, by round, I mean the following: 1 round means, compute something; send those values; and whatever the parties have sent in that round, receive them; that finishes 1 round.

Next round, process whatever messages you have received in the previous round as per the protocol; and then decide what to send in this round; send those messages. And other parties are also doing the same. So, they will be sending something; receiving the values sent by other parties. And then, round ends. So, like that, we have to count how many such send-receive compute are performed by the parties in this BGW protocol, assuming that the parties want to securely compute the linear function of n inputs.

So, where exactly interaction is needed in the BGW MPC protocol? The interaction is needed for secret-sharing the inputs of the parties. So, the first step of the protocol was, each party acted as a dealer and distributed shares for its respective input. So, for that, interaction is needed over the secure channel. If I want to implement this protocol; by interaction, I mean; say, I have to open the SSH socket or SSL socket and then I have to communicate my shares to the respective receiver over those channels.

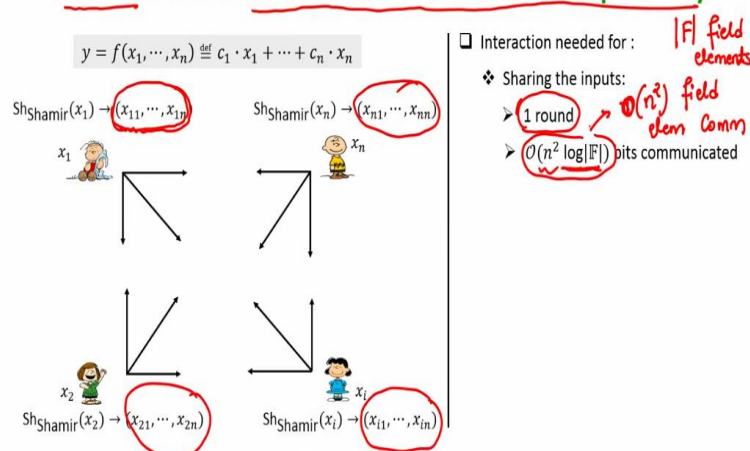
So, how many times I have to basically open the channel and communicate? That is what I am now trying to compute here. So, interaction is needed for sharing the inputs. And this can be

completed in 1 round. Even though when I demonstrated the protocol, I first demonstrated that P_1 shares, and then followed by P_2 , and then followed by P_3 , and then followed by P_4 ; but when you are implementing it, when P_1 is secret-sharing its input, at the same time, P_2 also can start its secret-sharing instance; because, there is no dependency between P_2 sharing its input and P_1 sharing its input; because the sharing polynomials which P_1 and P_2 pick, respectively, they are independent.

So, when P_1 is sending its share, at the same time, P_2 also can start sending its shares, because the channels are independent here. So, that is why, overall it will require 1 round of communication. It will not be the case that the sharing is happening here in a sequential fashion; no; the sharing is happening in parallel. So, that is why it requires 1 round. And how much communication happens if I count all the instances of sharing here? So, there are n instances of secret-sharing.

(Refer Slide Time: 21:12)

BGW MPC Protocol for Linear Functions: Round and Communication Complexity



And for each instance of secret-sharing, $\mathcal{O}(n)$ values or field elements have to be communicated. Each 1 field element has to be communicated to every other party, because that is a share. So, for 1 instance of secret-sharing, n field elements are communicated. For the second sharing instance, n field elements are communicated. For the i^{th} sharing instance, n field elements are communicated.

And for the last sharing instance, n field elements are communicated. So, overall, $\mathcal{O}(n^2)$ field elements are communicated. Of course, a party keeping its share of its own input to itself will not be considered as a communication; but in terms of order notation, the overall

communication that is happening is $\mathcal{O}(n^2)$. So, these many field elements are communicated, and each field element can be represented by these many bits.


Namely, there are these many number of field elements, and each field element can be represented by log of the number of elements in the field. So, since n^2 such field elements have to be communicated throughout, the communication needed for the sharing protocol, sharing part is this much. Now, where else is the interaction needed in the BGW protocol, if the function that needs to be communicated is a linear function?

(Refer Slide Time: 22:48)

BGW MPC Protocol for Linear Functions: Round and Communication Complexity

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


ShShamir(x_1) → (x_{11}, \dots, x_{1n})



x_1

y_1


ShShamir(x_n) → (x_{n1}, \dots, x_{nn})



x_n

y_n


y_2



x_2

ShShamir(x_2) → (x_{21}, \dots, x_{2n})

y_i



x_i

ShShamir(x_i) → (x_{i1}, \dots, x_{in})

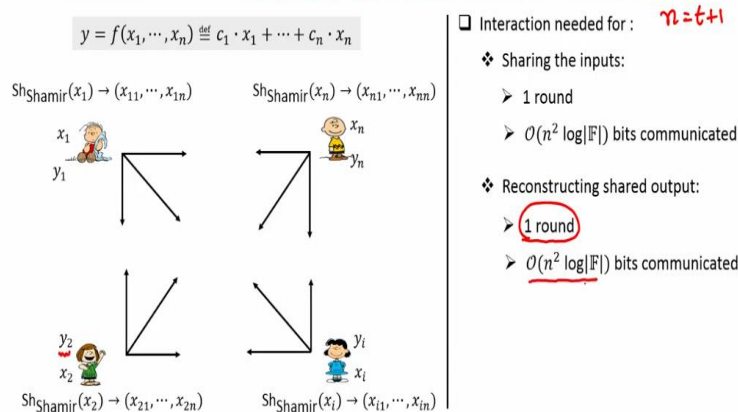
Interaction needed for : |F| field elements

- ❖ Sharing the inputs:
 - > 1 round ↪ $\mathcal{O}(n^2)$ field elem Comm
 - > $\mathcal{O}(n^2 \log |F|)$ bits communicated
- ❖ Reconstructing shared output:

Well, the shares of y , they are computed locally. That does not demand any interaction among the parties. But then, to learn the function output, the value y , the shares of y needs to be made public. Everyone has to announce the share y_i . So, if P_i is the i^{th} party, it has to announce the share y_i to every other party. It will be sufficient if any $t + 1$ of these n parties make their respective shares of y public; but in the worst case, my t could be all the way $n - 1$.

(Refer Slide Time: 23:25)

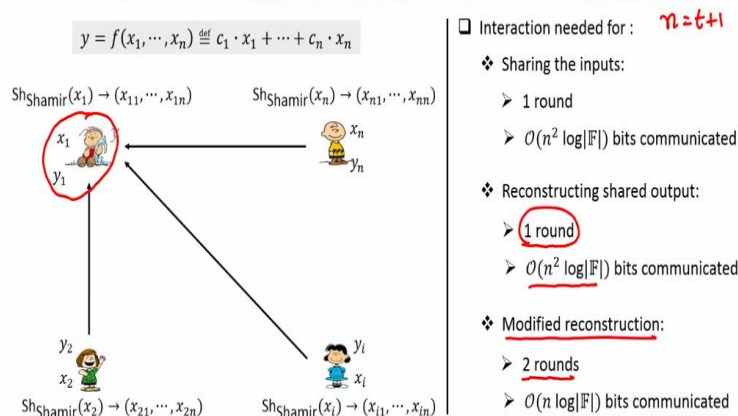
BGW MPC Protocol for Linear Functions: Round and Communication Complexity



That means, I may need all the n shares. So, that is why, when I am explaining here, I will say that, okay, all the n shares need to be public; but any $t + 1$ shares are sufficient if they are made public. So, there are 2 approaches here. One approach could be that, in a single round, every party sends its respective share of y to every other party. So, this will require a single round, and quadratic in the number of parties' communication. Alternately, I can do the following:

(Refer Slide Time: 24:04)

BGW MPC Protocol for Linear Functions: Round and Communication Complexity

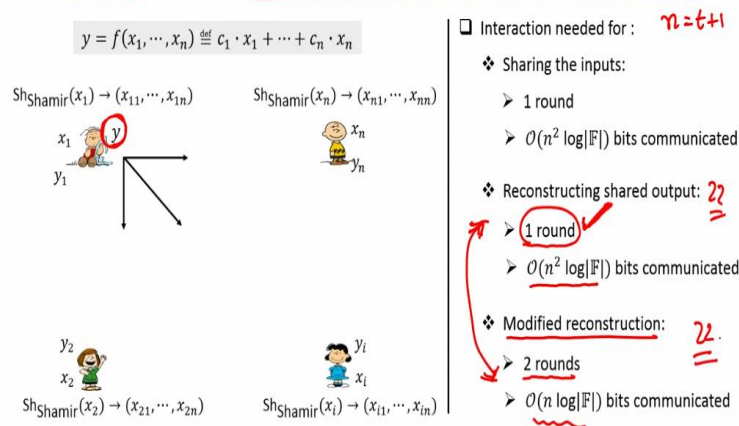


A modified reconstruction: Instead of 1 round, if you are willing to give me 1 additional round of communication, but ask me to save the communication, I can do the following: In the 1 round approach, every party sends its share of y to every other party. So, that is why, single round but n^2 communication; of course, $n^2 \cdot \log(\mathbb{F})$ bits. But if you give me 2 rounds, then what I can do is the following:

Let all the parties agree on a pre-determined party, as per the protocol itself. It could be any pre-determined party, any designated party; let it be the first party for simplicity. Then, in the first round, every party sends their respective share of y only to that designated party. So, that will require 1 round and $\mathcal{O}(n)$ communication in terms of field elements. That single party, now it will have all the shares of y , at the end of the first round.

(Refer Slide Time: 25:07)

BGW MPC Protocol for Linear Functions: Round and Communication Complexity



So, it can reconstruct y . And now, in the second round, it goes and announces the result y to everyone. So, that will be the second round. And how much communication happens in the first round? It is $\mathcal{O}(n)$, again in terms of field elements. How much communication happens in the second round? Again, a single field element, which is $\mathcal{O}(n)$. So, overall, $\mathcal{O}(n) \cdot \log(\mathbb{F})$ bits are communicated. So, there is a trade-off here.

If your network is kind of very slow, where you cannot afford to communicate multiple times among the parties, then go for this 1 round approach. That means, if the bandwidth is not an issue, but number of times the parties need to open the socket and interact is the issue, then go for this 1 round approach. But if the bandwidth is the issue, but the network is stable and you can communicate as many times as possible, but the restriction is that every time you are given a restriction that your bandwidth is very small, then go for this 2 round approach.

So, depending upon what is your primary constraint, whether it is the number of times you want to interact is more critical or whether it is how much you want to communicate every time, that is critical; you decide whether you want to go for this 1 round reconstruction approach or whether you want to go for this 2 round reconstruction approach.

(Refer Slide Time: 26:41)

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, these are the references used for discussing today's lecture. We have not yet seen the security analysis. We have seen only the protocol details for computing the linear function, and we have analysed the round and communication complexity. Thank you.