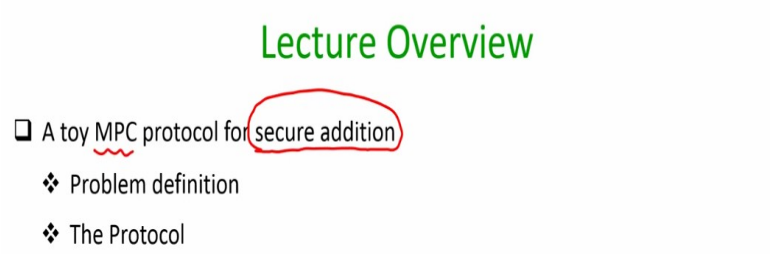
Secure Computation - Part I
Prof. Ashish Choudhury
Department of Computer Science
International Institute of Information Technology, Bangalore

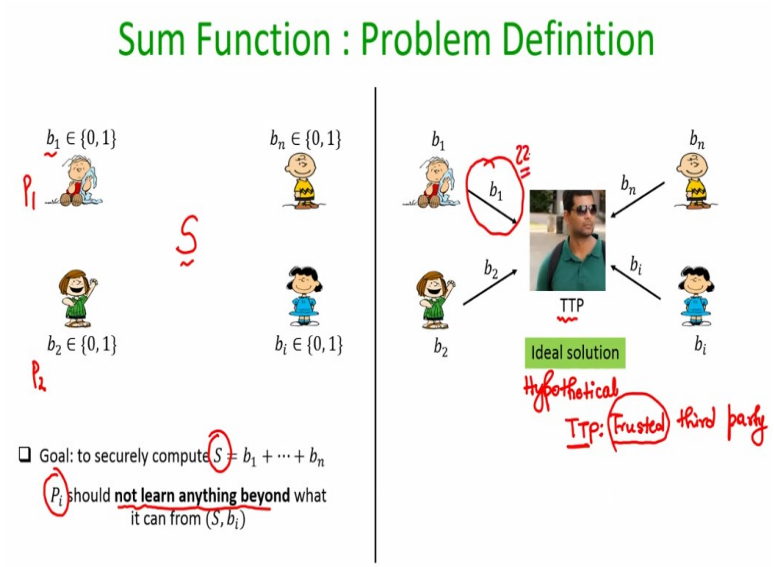
Module - 3
Lecture - 16
A Toy MPC Protocol

(Refer Slide Time: 00:35)



Hello everyone. Welcome to this lecture. So, the plan for this lecture is as follows: So, now, we will start our discussion on multi-party computation, how to design protocols for performing secure computation. And in this lecture, we will do a warm up. We will see a very simple function, namely that of secure addition; we will see the problem definition, what exactly we mean by secure addition; and we will see a very simple MPC protocol for securely computing the addition function.

(Refer Slide Time: 01:06)



So, here is the definition of the addition function, which I also call as the sum function. So, you are given set of n parties, P_1, P_2, P_i, P_n, and each party has a private bid. So, party 1 has the bid b_1, party P_2 has the bid b_2, ith party has the bid b i, nth party has the bid b n, and they are private, but everyone knows that the input of every party is either 0 or 1. That is a public information, but what exactly is the value of the bid, that is not known.

And the goal is to compute the addition of the n bids. So, let us denote the sum as S. So, the parties want to learn the sum of the n bid values. That is the goal here. And if you are wondering what is the bid deal, why cannot every party just announce that this is my bid, and then let everyone take the sum of the announced bids? Well, to make the problem interesting, since we want to do secure computation, the requirement here is that, we want to learn the sum, and in the process, it should be ensured that no party P_i should learn anything beyond what it can learn from the sum and its own input.

That means, what I want here is to ensure the following: The sum S will be finally learnt by everyone. Now, P_1, it will have its bid b_1, and it will learn the final sum S. From that, it can infer something regarding the inputs of the remaining parties. Only that much should be inferred in the whole protocol. Similarly, if I consider b_2, P_2, it will have its own bid and it will learn the final sum.

Based on that, it will have some knowledge about the inputs of the remaining parties; that is allowed. Other than that, nothing additional should be revealed, and so on. That is the goal. So, now that simple protocol where all the parties just announce their bids in public, and then

you take your sum, it is not going to work; because you are not only learning the sum, you are learning every other party's input bid, which is not supposed to be learnt.

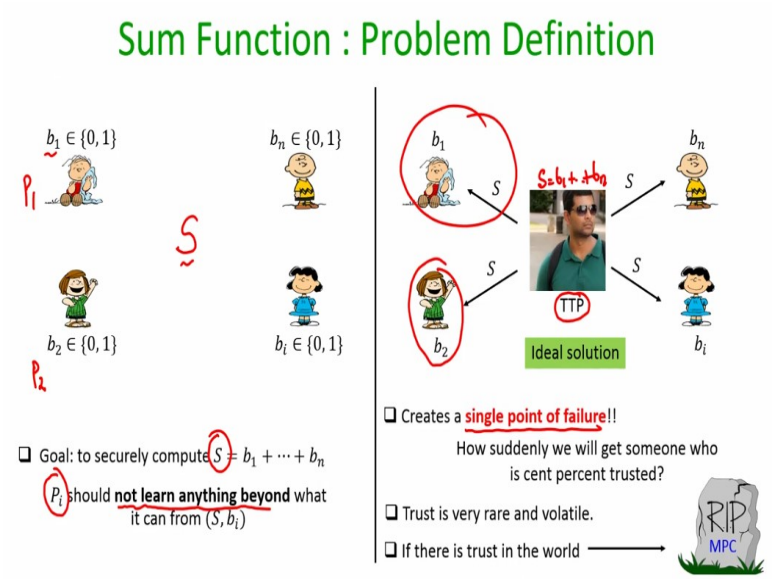
So, now, let us try to understand what exactly it means by saying that P_i should not learn anything beyond the final result and its own input. What does it mean, does not learn anything beyond? So, let me propose an ideal solution. This is a hypothetical solution. That means, there are problems with this solution, but let us consider this solution and try to understand that what is the information a party learns in this ideal solution.

In this ideal solution, we assumed that we have what we call as a trusted third party, TTP. That means, it is a party different from these n parties. And it is trusted in the sense, it is assumed that he is the friend of all the n parties; friend in the sense that he is willing to help the n parties compute the sum, but even if he learns the input of the respective parties, he is trusted, he will not be leaking those input values to the other parties.

That is the guarantee, that is the assumption, that is what we mean by a trusted entity, equivalent to God, you can imagine. So, assume that such a trusted party is present. And imagine that there is a mechanism for each party to privately communicate with this trusted third party. That means, there is some dedicated channel between every party and this trusted third party. Then, what will be the mechanism to securely compute the sum function?

Well, the simple protocol will be, every party individually communicate its private input to this trusted third party. And this communication is happening over the private channel, which is available between each party and the trusted third party. That means, when b_1 is communicated from P_1 to this trusted third party, no other party can make out what is the bid, what is the value of b_1 that is communicated and so on. That is the assumption here.

(Refer Slide Time: 05:55)



Now, once this trusted third party learns all the n input bids, it can itself compute the sum, because it has got all the n input bids. And then, it simply announces the result to everyone publicly. That is the ideal solution. And why it is called ideal solution? Because this is the most secure protocol that you can think of. In this process, what is the information any party learns about other parties' input? Nothing.

There is only 1 communication happening between every party and the TTP. The party sends input, gets back the output; that is all. If I consider say for instance, party P_2, will it learn anything about b_1, b_2, b_i, b_n, when they are communicating to the TTP? The answer is no, because that communication is happening over secure channels. And anyhow, S is learnt by everyone.

So, this is the most secure solution that you can think of for computing the sum function. And in this protocol, it is indeed ensured that each party, if you consider each individual party, it just has finally its own input and the sum value. From that, whatever they can infer regarding other party's input; they can make some hypothesis, okay, it could be the case that the inputs of this combination of parties is this, blah, blah, blah; all these things are finally anyhow allowed to be revealed.

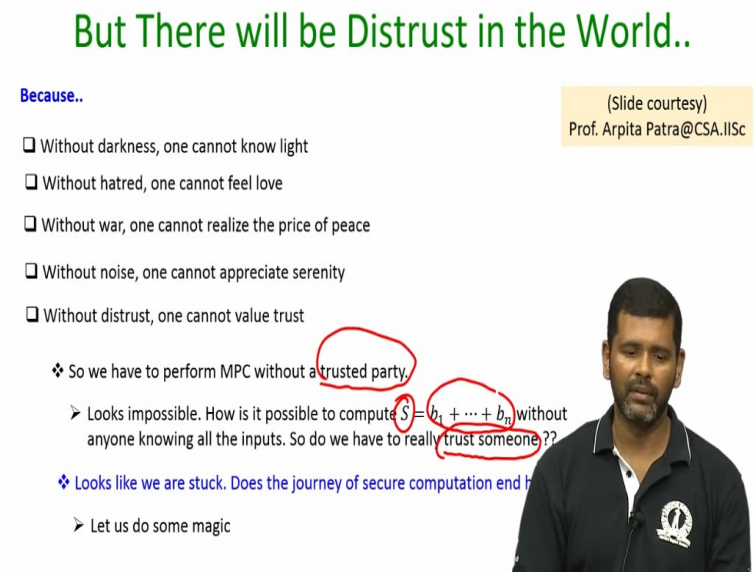
We cannot prevent that from getting revealed from the sum and the individual inputs, but since no communication is happening among the parties here, in this ideal solution; that is important; no communication is happening among the parties in this solution. Individually, a

party is not allowed to learn anything in this ideal solution. But the problem with this ideal solution is that, it creates what we call as single point of failure.

Namely, how suddenly we will get someone who is cent percent trusted? This whole solution works under the assumption that there is a third party, a friend party who is trusted by everyone. How in this real world you can find someone who is indeed cent percent trusted? Because trust is very rare and volatile. And if there is trust in the world, then we do not need to design, we do not need to study MPC.

Because, if a trusted third party is there, then whoever are the set of n mutually distrusting parties, they can take the help of this trusted party to keep the privacy of their messages, inputs, and at the same time perform computation on their inputs. But at the first place, we are actually trying to design a protocol because we know that there is no one called a trusted third party in this real world.

(Refer Slide Time: 08:54)



Because, there is always a distrust in the world; because, without darkness, one cannot know light; without hatred, one cannot feel love; without war, one cannot realize the price of peace; without noise, one cannot appreciate serenity; and without distrust, one cannot value trust. So, the ideal solution that we have proposed, that is a hypothetical solution, that will work only under the assumption that you have someone who is trusted by everyone.

So, now it looks like we are kind of stuck. We do not have any trusted third party, we just have the n parties, that is all. And at the same time, we want a mechanism which allows those

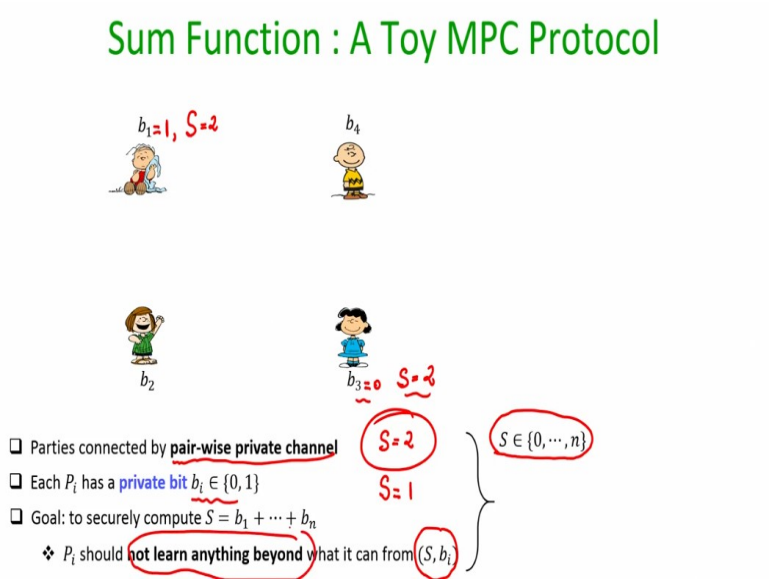
n parties to maintain the privacy of their respective bids; but at the same time, the sum should be computed and announced publicly. Looks like an impossible task, because how can it be possible that without anyone knowing all the individual bids, b_1, b_2, b_n, the sum is computed?

Because, if at all the sum needs to be computed, the values of the n bids have to be known. Who should know that? Because we cannot afford to let P_1 learn all the n bids, compute the sum and announce; or we cannot let P_2 know all the n bids, compute the sum and announce; because our goal is that, no single party should learn anything additional beyond its own input bid and the final sum.

So, you might be now wondering that, do we indeed require a trusted third party to compute this very simple sum function? We are talking about the real world MPC motivation examples that we talked about. They are very complex functions; privacy-preserving data mining, checking whether there is a probability of 2 satellites colliding and so on. Forget about those complex functions, we are just talking about a very simple sum function, that is all.

How can it be possible without trusting anyone, any single party, still we are able to compute the sum of n bid values? Looks like an impossible task, but we will do now some magic. We will see a very simple protocol for computing this sum function.

(Refer Slide Time: 11:18)



So, now, we do not make any assumption that we have a trusted third party. We will be designing a protocol according to which the parties will communicate messages among

themselves. And we assume that to communicate, parties have pairwise private channel. That means, there is a way for every party to communicate some message, whatever message it wants to communicate to any other receiver, in a private, in a perfectly secure way.

And we had seen in the last lecture, how exactly such channels can be realized. The problem setting is the following: Each party has a private bid b_i, and the goal is to securely compute and announce the sum of n bid values. And it should be done in such a way that no party P_i should learn anything beyond what it can learn from the final sum and its own input bid during the protocol execution.

That means, whatever messages any party P_i learns, gets during the protocol execution, that should be kind of independent of the inputs of the other parties. That is what roughly we want to achieve here. So, what will be the protocol? So, to begin with, it is easy to see that the final sum is in the range 0 to n; 0, because it could be the case that the inputs of all the parties is 0; n, because it could be the case that all the input parties have the bid 1.

By the way, if all the parties have the input bid 1, and if the final sum is learnt, is to be n, then anyhow, parties will learn that all the parties have participated with input 1. That is not a breach of privacy, because that is allowed to be learnt, remember. If the final sum is n, everyone will know that, okay, every party has participated with input 1; that is leaked anyhow, from the input and the final output.

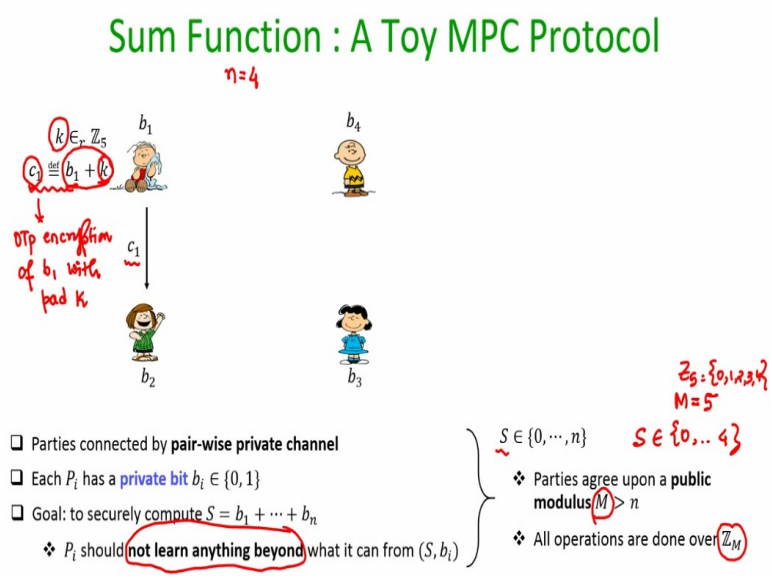
That is not called a privacy leak or privacy breach here. In the same way, if the final sum turns out to be 0; fine; everyone will learn that every party participated in the protocol with input 0; that is not a privacy breach. But suppose if the sum is turned out to be, say 1, then again, the party whose input is 1, he will know that, okay, I only have participated with input 1, but others have participated with 0; fine; that is again not a privacy breach.

But suppose if sum is turned out to be 2 after the execution of the protocol and suppose b_1 is equal to 1, then party 1 will learn from this output that, okay, definitely there is only 1 more party whose input is 1. Which party has participated with input 1? That should not be learnt, because that is not allowed to be learnt from b 1 equal to 1 and s = 2. The information that is allowed to be learnt from b 1 equal to 1 and s = 2 is that there is another party and only 1 party who has participated with input 1.

That is allowed to be learnt, but anything additional apart from that, that should not be learnt in the protocol execution. Whereas, if b_3 is equal to 0 and s = 2, from its viewpoint, it will be learning that anyhow that there are 2 parties who have participated with input 1; that is allowed to be learnt from b_3 equal to 0 and s equal to 2. But who are those 2 parties? That should not be leaked.

So, that is what roughly I mean here that nothing additional should be revealed beyond a party's own input and the function output. So, do not consider that if s = n, then everyone learns that all the parties have participated with input 1, and that is a privacy breach; no. That is why I have stressed here the term anything beyond.

(Refer Slide Time: 15:27)



So, since the sum is going to be a value in the range 0 to n, the parties will agree upon some public modulus capital M. Agree upon means, they will, the protocol will be designed in such a way that all the operations will be performed modulus M. That means, it will be ensured that all the values computed in the protocol, they lie in the range 0 to M - 1; and M is strictly greater than n. That is what we mean by parties agree upon this public modulus.

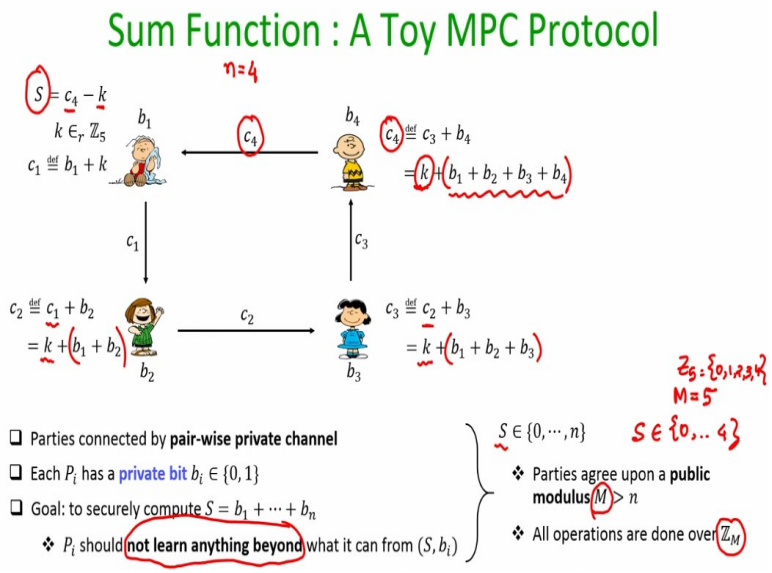
Now, here is the protocol. To start with, the first party, it picks a random value; and since here n = 4, I am taking for demonstration purpose 4 parties here; that is why my sum value could be in the range 0 to 4; and that is why we have agreed upon the, we will be performing all the operations modulo 5. And Z_5 means, the set 0, 1, 2, 3, 4. So, what P_1 does is the following:

It picks a random value k from this set Z_5; with equal probability, it could be 0, 1, 2, 3, 4; and that is known only to P_1. Now, it computes an encryption of its bid b_1. And how it computes the encryption? It simply adds the value k to its bid, and that is the ciphertext c_1, which it now communicates to the second party over the private channel which is available between P_1 and P_2.

So, people who are familiar with one-time pad encryption, this c_1 is nothing but a OTP encryption of b_1 with pad k. And remember, this plus operation is performed modulo 5. That means, the value of c_1 will remain in the range 0 to 4, because every time a value crosses 4, we will perform the mod operations. And P_2 will know that, okay, whatever value it is receiving, that is an OTP encryption of the bid b_1.

But since the pad k is uniformly random, it does not reveal anything about the actual value of the bid P_1. Now, what P_2 does? It takes the OTP encryption c_1 and add its own input bid to that ciphertext.

(Refer Slide Time: 18:44)

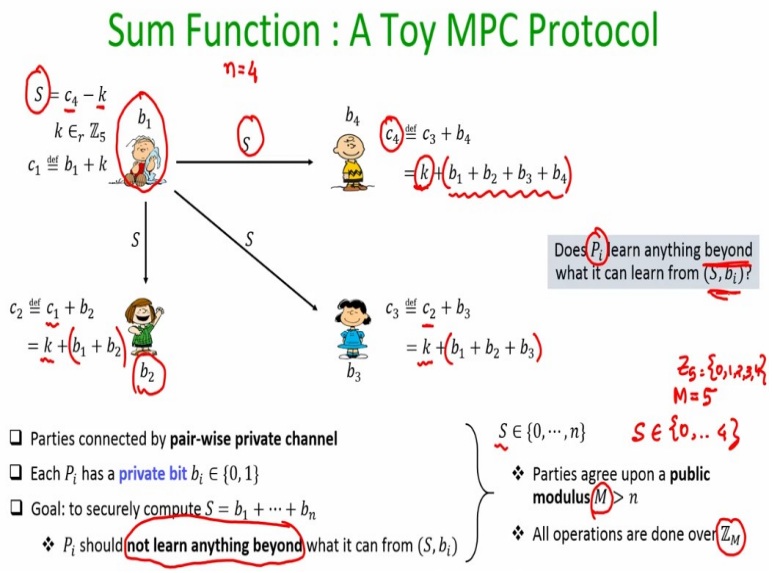


And if we expand the OTP ciphertext c_1, the ciphertext to c_2, which now P_2 is communicating to the next party, can be interpreted as if the sum of the first 2 bids, b_1 and b_2 is encrypted using the pad k. And this ciphertext c_2 is communicated over the private channel available between the second party and the third party. And now, the third party further extends this process.

It takes the ciphertext which it is receiving and it adds its own input bid. And now, this will be treated as an OTP encryption of the sum of the first 3 input bids using the pad k. And this will be communicated to the fourth party. And now fourth party adds its own input bid and this will be treated as if the sum of the 4 input bids is encrypted using the pad k. And this ciphertext or this encryption will be communicated back to the first party.

Now, first party; if you see the value of c_4 closely here; is getting an OTP encryption of the sum of 4 bids with respect to the pad k which is available with P_1. So, it can just remove the pad k or it can actually perform the decryption of your ciphertext c_4. So, the decryption operation will be unmasking the pad. Unmasking the pad means, subtracting out the pad. So, if the pad k is subtracted from this OTP encryption c_4, that will give the sum S, namely, the summation of b_1, b_2, b_3, b_4, which this first party can now announce to everyone publicly.

(Refer Slide Time: 20:46)

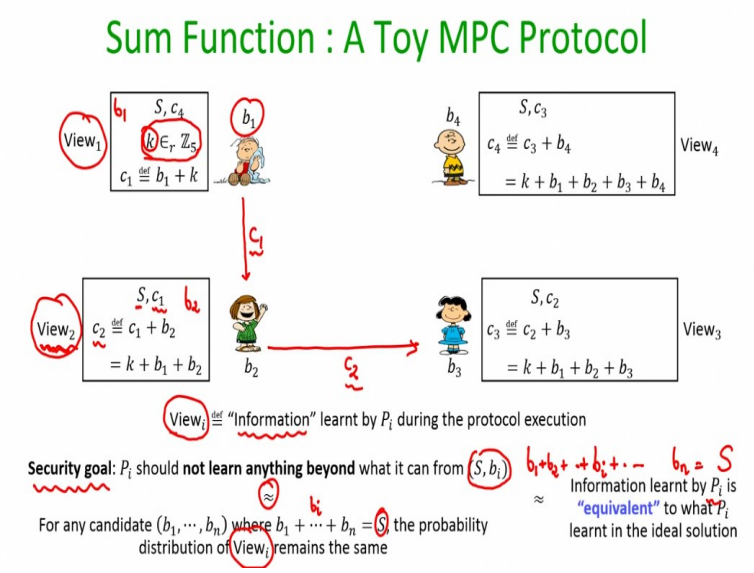


That is a simple protocol. Now, we have to analyse this protocol and we have to argue that in this protocol, any single party, we call it P_1 or P_2 or P_3 or P_4, any single party, does it learn anything beyond what it can learn from S and b_i, even if it is computationally unbounded? Let us make b_i computationally unbounded. That means, even if the first party is computationally unbounded, does it learn anything about the inputs b_2, b_3, b_4, beyond what it can learn about b_2, b_3, b_4, from the sum and its own input value b_1?

That is what we want to argue. In the same way, we want to argue that suppose if P_2 is computationally unbounded, after the protocol gets over, he will anyhow learn the sum, he has his own bid value b_2. Now, it wants to analyse the messages that it has received during

the execution of the protocol. Is it possible that if P_2 is computationally unbounded, there is a mechanism for P_2 to analyse the messages that it has received and infer anything additional about b_1, b_3, b_4, beyond what it can infer from the final sum and b_2? and so on. And we want to perform this analysis for every single party here.

(Refer Slide Time: 22:21)



So, to prove the privacy property that indeed this simple protocol satisfies the privacy property, what we will do is the following: We will define first what we call as the view of any party P_i. So, I have written down the view of every party in this rectangular box. And roughly, this is the information which is learnt by the party P_i, during the protocol execution. So, the view will have definitely the inputs of the parties.

So, sorry, I have not written it down explicitly here in the boxes. So, view will definitely have the inputs of the parties. And if there are any random coins which are used by the parties during the protocol execution, they also will be a part of the view. So, if you see here, no party except the first party, tosses a random coin. It is only the first party who picks the random pad, but apart from that, no party has any random coin being used in the protocol.

So, k also will be a part of the view of the first party. And then, whatever messages that party has received from other parties and communicated to the other parties, they will be a part of the view of that party. So, for instance, if I consider this party number 2, his view will be, it is learning the final sum, it would have received the ciphertext c_1, so, that is why c_1 is part of its view.

And it would have communicated c_2 to the other party, so, that is why c_2 is a part of its view, and b_2 also will be a part of its view, because that is his own input. And to prove that indeed this protocol is secure, this simple toy MPC protocol is secure, we have to argue that P_i should not learn anything beyond what it can learn or compute itself from S and b_i. And this is equivalent to saying that, anyhow the sum S will be learnt and b_i is available with the party P_i, based on b_i and S, it will have multiple candidate values for the inputs of the remaining parties, which along with b_i would give the sum S, which is learnt by the party.

We have to show that this $View_i$ which is learnt by the party P_i during the protocol, that is equiprobable with respect to every such candidate b_1, b_2, b_i, b_n, which sums up to the given S. And remember here that the view V_i here, namely, the view of the each party here, it is not a fixed set of values. That means, it is not the case that every time the protocol is executed, even with the same set of inputs b_1, b_2, b_3, b_4, the same ciphertext c_1, c_2, c_3, c_4, would be circulating in the system.

Because, c_1 will be different every time, because the pad, one-time pad is picked randomly in each execution. So, even if b_1 is same across multiple executions, c_1 will take different values. And if c_1 takes different values, c_2 will take different values, even if b_2 is fixed, and so on. So, that is why, the contents of $View_i$ is not going to be a fixed set of contents, even if the inputs b_1, b_2, b_3, b_4 are fixed. And that is why, $View_i$ here is a random variable, and it can take different values with different probability.

We have to argue here for privacy that the probability distribution of $View_i$ is independent of the inputs of the other parties. Namely, you fix b_i, you fix the sum S, then for every candidate remaining inputs, which along with this fixed b_i gives you this fixed S; $View_i$ could occur with equal probability. And if we show this, then this is equivalent to showing that the information learnt by P_i is more or less equivalent to what P_i would have learnt in the ideal solution.

Remember, in the ideal solution, the view of the ith party would have been just its own input and the function output, namely the sum S, because that is the only information learnt by P_i in that ideal solution. But in this toy MPC protocol, $View_i$ has other things as well, the ciphertext received by the party and the ciphertext which it communicates to the other party and so on.

If we show that view is distributed independent of the inputs of the other parties, then that is equivalent to showing that basically $View_i$ has the same amount or equivalent amount of information which P_i would have learnt in the ideal solution. And that is what we will do in our next lecture. Thank you.