


**Secure Computation: Part II**  
**Prof. Ashish Choudhury**  
**Department of Computer Science and Engineering**  
**Indian Institute of Science, Bengaluru**


**Lecture - 25**  
**Domain Extension for Perfectly-Secure Broadcast Based on RS Error-Correcting Codes: III**

(Refer Slide Time: 00:25)

## Lecture Overview

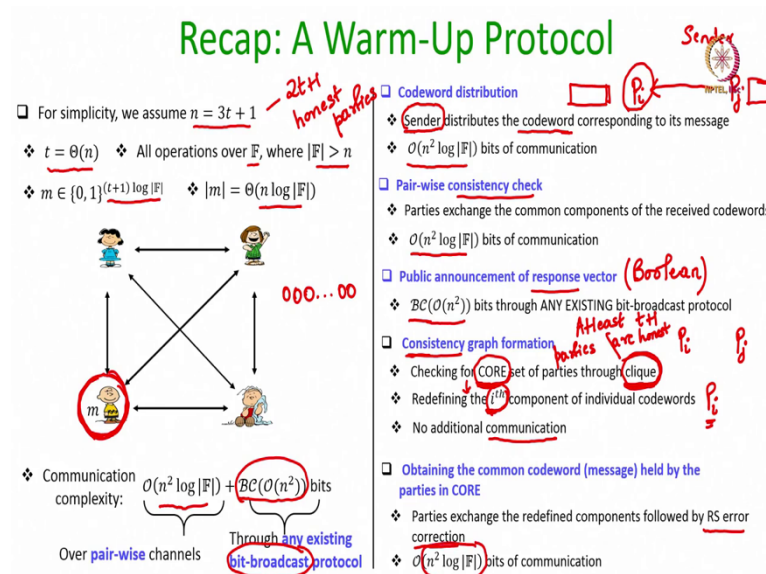


- ❑ Domain extension for perfectly-secure broadcast
  - ❖ An exponential time protocol



Hello everyone. Welcome to this lecture. So, in this lecture we will continue our discussion regarding the efficient domain extension protocol for perfectly secure broadcast. So, we will use the warm up protocol, that we had seen in the last lecture and using that we will see the actual domain extension protocol. The domain extension protocol that we will discuss in today's lecture will have exponential running time, exponential computation time. And, later on we will see how we can modify the protocol so, that the its running time becomes polynomial time.

(Refer Slide Time: 01:02)



So, just a quick recap regarding the warm up protocol. So, in that protocol we assumed  $n = 3t + 1$ , that is just for simplicity, so that we can use the fact that  $t = \Theta(n)$  asymptotically, during complexity analysis. In that protocol all operations are performed over a finite field  $\mathbb{F}$ , whose cardinality is at least  $n + 1$ . In the protocol there is a designated sender with some message  $m$  whose size is  $(t + 1) \log |\mathbb{F}|$  bits.

And, asymptotically we can say that since  $t = \Theta(n)$ , that means, the sender's message is of size  $\Theta(n \log |\mathbb{F}|)$  bits. And, we had seen the warm up protocol where it is guaranteed that all the parties identically receives the sender's message, even if the sender is potentially corrupt. And, the communication complexity of that protocol has two parts. The communication done over the pair wise channels and which does not require any invocation of the existing bit broadcast protocol.

So, that part of the communication is  $n^2 \log |\mathbb{F}|$  bits and in addition the protocol also requires the parties to broadcast total of order of  $n^2$  bits using any existing reliable bit broadcast protocol. So, that part of the complexity is denoted by the notation  $BC(n^2)$ .

So, let me quickly go through the warm up protocol, the various stages. So, the first stage was the code word distribution where sender converts its message  $m$  into a message consisting of  $t + 1$  elements from the field. And, corresponding to that it computes a Reed-Solomon code word.

And, in order to send its message to every party, it actually sends the code word corresponding to its message to all the parties. And, this step requires a communication of  $n^2 \log |F|$  bits over the point to point channels. If the sender is honest, not under the control of the adversary, then it will send the identical code word to all the parties. But, it is not guaranteed that sender is always honest, sender also could get under the control of the adversary.

So, in the rest of the protocol the parties verify whether the sender has sent a common code word to sufficiently many parties. And, if it has sent a common code word to a sufficiently many parties then that code word is somehow get transferred to all the parties, who may not have received the same code word from the sender if sender would have been honest.

So, this happens through various steps. So, the second stage of the protocol was the pair wise consistency check, where every pair of parties exchange common components of the received code words ok. So, if there is a sender, then it would have sent some code word to  $P_i$  and some code word to  $P_j$ . Then, ideally if the sender is honest and if  $P_i$  and  $P_j$  are also honest, then  $P_i$  and  $P_j$  should have received the same code word from the sender. To verify the same, every pair of parties  $P_i$  and  $P_j$  exchange the supposedly common components of the received code words.

So, they are not exchanging the entire code words because that will blow up the communication complexity. Rather every pair of parties exchange only two common components, two supposedly common components of their received code words. And, this requires a communication of  $n^2 \log |F|$  bits over the point to point channels. The third stage is public announcement of the pair wise consistency check. And, it is this particular step which requires the invocation of existing bit broadcast protocol.

Namely, in this stage what happens is every party  $P_i$ , once it receives the supposedly common components from  $P_j$ , it checks those components against its own components. And, if they match then  $P_i$  is ok with  $P_j$ , otherwise  $P_i$  says that I am in conflict with  $P_j$ . So, whether  $P_i$  is in conflict with  $P_j$  or not, that is captured through a response vector which is a Boolean response vector, where an entry will be 1 against  $j$ , if  $P_i$  is fine with  $P_j$  and the entry will be 0 corresponding to  $P_j$ , if the pair wise consistency fails.

And, that vector now needs to be made public. And, to make that vector public each  $P_i$  will make public its response vector by using any bit broadcast protocol, where  $P_i$  itself now acts as a sender and its message will be its response vector and bit by bit it broadcast the entire response vector, so that all the parties receive an identical copy of  $P_i$ 's response vector ok.

So, this stage will require a total broadcast of  $n^2$  bits through any existing bit broadcast protocol. Now, based on the response vector which are now publicly available, the parties construct a consistency graph to check which pair of parties are fine with each other with respect to the supposedly common components of their respective code words. Namely, we add an edge between the node representing party  $P_i$  and the node representing party  $P_j$ , if  $P_i$  has no conflict with  $P_j$  and  $P_j$  has no conflict with  $P_i$  in their respective response vectors. And, then we check whether sufficiently many parties are pair wise consistent with each other. Namely, we check whether there exist a clique of size at least  $2t + 1$  which should be present in the consistency graph, if sender has behaved honestly. Because, if sender has behaved honestly then it should have sent the common code word to all the honest parties in the system and there are at least  $2t + 1$  honest parties guaranteed in the system.

So, those  $2t + 1$  parties will definitely constitute a clique. Of course, there could be more than  $2t + 1$  parties who can be pair wise consistent. But, if at all sender is honest then it should be guaranteed that at least  $2t + 1$  parties are pair wise consistent. So, that is what the parties check by checking whether there exist a clique of size at least  $2t + 1$ . Clique means a complete graph of size at least  $2t + 1$ .

If such a clique exists then the set of clique parties is considered as the CORE set of parties. And, in that clique at least  $t + 1$  parties are guaranteed to be honest, at least  $t + 1$  parties are honest in that clique. Of course, not all the honest parties necessarily be a part of the clique, that can happen say for instance if the sender is corrupt and it has sent an identical code word to only  $t + 1$  honest parties in the system and the rest of the honest parties in the system have got a different code word.

Even in that case it may so happen that we get a clique of size  $2t + 1$ , because the corrupt parties may say in their response vector that they are not in conflict with the honest parties. So,  $t + 1$  honest parties along with  $t$  corrupt parties can constitute a clique of size  $2t + 1$ , that also is one possibility. If a CORE is not found then that definitely implies that the sender is corrupt and the parties stop the protocol there itself.

And, they take a default message on the behalf of the sender assuming that sender wanted to broadcast a string consisting of  $(t + 1) \log \log |F|$  number of 0s.

However, if a clique is obtained, that guarantees that all the honest parties in CORE have received a common code word. And, then the goal will be to somehow ensure that even the honest parties outside CORE, if there exist any such honest party, they should also get the same code word.

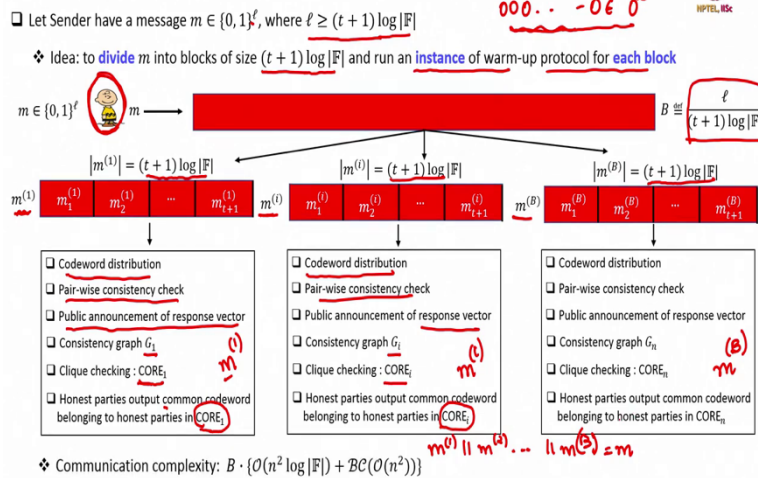
So, for this in the same stage, the parties redefine the  $i$ th component of their received code words. They forget what exactly was the code word they would have received from the sender. They redefine their  $i$ th component based on whatever communication they had from the parties in CORE during the pair wise consistency check, that will ensure that each  $P_i$  gets the  $i$ th component of the common code word held by the parties in CORE.

And, this stage does not require any communication because everything happens locally on the consistency graph. And, then the last stage is where every party  $P_i$  exchanges its redefined component of the common code word held by the parties in CORE. And, then the parties apply the Reed-Solomon error correction on the redefined components to ensure that every party whether they are part of code or not, irrespective of that they get the common code word held by the honest parties in CORE.

And, this stage requires a communication of  $n^2 \log \log |F|$  bits over the point to point channel. So, that was the warm up protocol.

(Refer Slide Time: 11:55)

## An Exponential Time Domain Extension Protocol



Now, let us see that how we can plug in that warm up protocol to get a domain extension protocol. In our domain extension protocol, sender will have a bigger message now. Its message will be consisting of  $l$  bits, where  $l$  is at least  $(t+1) \log \log |F|$ , ok. Now, the sender's goal is to send this message identically to all the parties and it should be guaranteed that even if the sender is potentially corrupt, all the honest parties at the end of the protocol receive an identical message output, an identical message on the behalf of the sender.

It should not happen that if the sender is corrupt then an honest  $P_i$  outputs one string of length  $l$  bits and another honest party  $P_j$  outputs a different string of length  $l$  bits, that should not happen. So, to do that sender does the following. It divides its message  $m$  into chunks of size  $(t+1) \log \log |F|$  ok. And, for each such block, each such chunk, it runs an instance of the warm up protocol that we had discussed till now. So, the first chunk of  $(t+1) \log \log |F|$  bits, suppose I represent its components as  $m_{1,1}, \dots, m_{1,t+1}$ .

So, the first block of  $(t+1) \log \log |F|$  bits, it is further converted into  $t+1$  field elements. Like that, the  $i$ th block of sender's message will have  $(t+1) \log \log |F|$  bits, which are converted into  $t+1$  field elements. And, like that the  $B$ th block where  $B = \frac{l}{(t+1) \log \log |F|}$ , we will have those many blocks in sender's message, those many blocks of size  $(t+1) \log \log |F|$  bits.

So, we have the message blocks  $m^{(1)}, \dots, m^{(B)}$ . And for each such block, sender will be running one instance of the warm up protocol. So, for instance for the first block sender will convert the  $t + 1$  field elements in its block into a code word by treating the message block  $m^{(1)}$  as the input for Reed-Solomon encoding algorithm. The code word will be distributed. The parties will perform the pair wise consistency check. Then, the parties will publicly announce the response vector. A consistency graph will be formed. It will be checked whether a clique of size  $2t + 1$  is there or not. If it is there then the code word components will be redefined and then the parties will somehow ensure that through the help of the redefined code word components, all the honest parties end up getting the code word held by the honest parties in CORE 1.

In parallel for the  $i$ th block the same computation will be performed; code distribution, pair wise consistency check, announcement of the response vector, consistency graph formation, CORE formation, CORE checking. And, then if the CORE is present then ensuring that the common code word held by all the honest parties in CORE $_i$  gets transferred to all the honest parties.

And, in parallel a similar computation will be performed for the  $B$ th block through  $B$ th instance of the warm up protocol. Now, if any of these  $B$  instances of the warm up protocol a clique is not obtained, that automatically implies that the sender is corrupt. And, we can discard the sender for the overall protocol, for the overall domain extension protocol and we can take a message consisting of  $l$  number of 0s on the behalf of the sender as the output.

Of course, an honest sender will never be discarded because, an honest sender sends an identical code word in each of the individual instances of the warm up protocol. So, for each individual instance of the warm up protocol, a clique of size  $2t + 1$  will be obtained ok. And, it is easy to see that this protocol will satisfy the termination property because the warm up protocol terminates.

So, each of the instances of the warm up protocol terminates. This extension protocol also satisfies the validity property, because if sender is honest then through the first instance of the warm up protocol the output of all the honest parties will be  $m^{(1)}$ , in the  $i$ th instance of the protocol, the output of all the honest parties will be  $m^{(i)}$  and in the  $B$ th instance of the warm

up protocol, the output of all the honest parties will be  $m^{(B)}$ . And, the overall output of all the parties will be then  $m^{(1)} || m^{(2)} \dots || m^{(B)}$ , which is nothing, but the sender's message.

So, validity is satisfied and consistency is also satisfied. Well, if the sender is discarded then consistency is trivially satisfied, because all the honest parties will be outputting a string consisting of  $l$  number of 0s. But, if the sender is not discarded then at the end of the first instance of the warm up protocol, all the honest parties will have a common output consisting of  $(t + 1) \log \log |F|$  bits.

In the  $i$ th instance of the warm up protocol, all the honest parties will have a common output. And, through the  $B$ th instance of the warm up protocol, all the honest parties will have a common output. And, the overall output of all the parties is a concatenation of the individual outputs obtained from the individual instances of the warm up protocol. So, consistency is also satisfied.

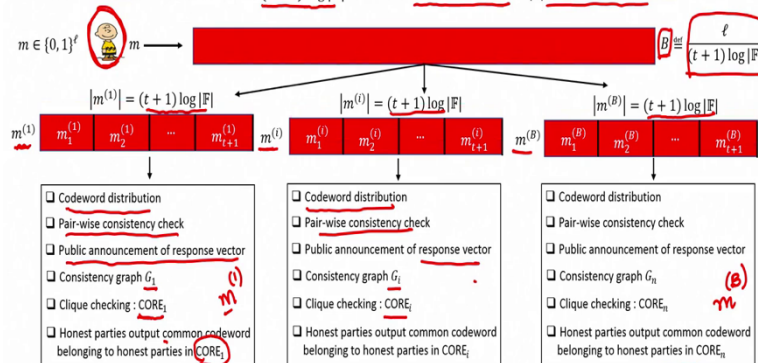
But, now let us see how much communication complexity this domain extension protocol will have. So, the complexity of this domain extension protocol will be  $B$  times the cost of one instance of the warm up protocol.

(Refer Slide Time: 19:13)

## An Exponential Time Domain Extension Protocol

Let Sender have a message  $m \in \{0, 1\}^\ell$ , where  $\ell \geq (t + 1) \log |F|$

Idea: to divide  $m$  into blocks of size  $(t + 1) \log |F|$  and run an instance of warm-up protocol for each block



Communication complexity:  $O\left(\frac{\ell}{(t + 1) \log |F|} \cdot n^2 \log |F|\right) + BC\left(\frac{\ell}{(t + 1) \log |F|} \cdot n^2\right) = O(n\ell) + BC\left(\frac{n\ell}{\log |F|}\right)$

And what is the cost of one instance of the warm up protocol? By substituting that value and by substituting the value of  $B$ , namely the number of blocks that we have here or the number of instances of the warm up protocol that we are running here, the overall cost turns out to be



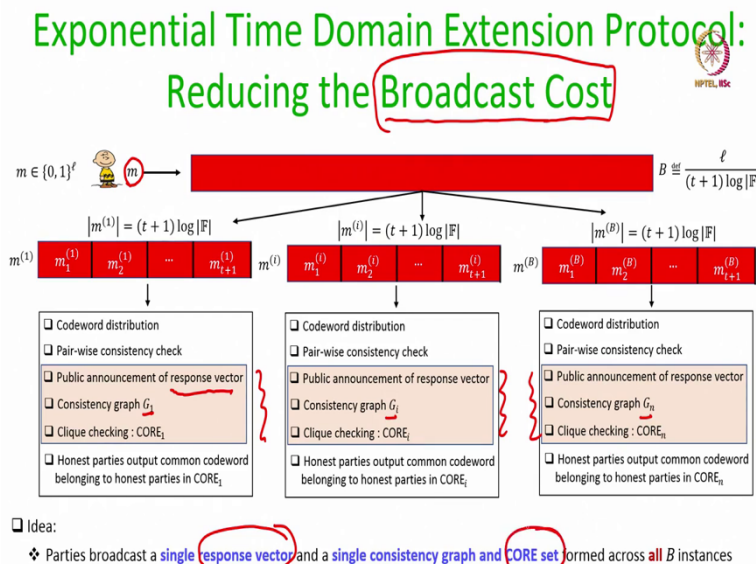
this much. Now, remember that  $t + 1$  is asymptotically  $\Theta(n)$ . So, this  $t + 1$ , I can treat as  $\Theta(n)$ . So, one  $n$  gets cancels out in the numerator.

So, this part of the communication becomes  $n$  times  $l$ , that is fine that is very good; over the point to point channel  $n$  times  $l$  bits of communication is involved. But, how much broadcast is needed through existing bit broadcast protocol, that is not independent of  $l$  that depends upon  $l$ . This factor, the amount of broadcast required to be done through the existing bit broadcast protocol that is not independent of  $l$ , that depends upon  $l$ .

And, that loses the whole purpose of the domain extension. The whole purpose of the domain extension protocol was that we wanted to have a broadcast protocol, where the number of instances of the bit broadcast is independent of the sender's message namely  $l$ . But, that is not happening right now ok.

So, even though it looks like that we are very close to the goal of achieving a communication complexity of  $n$  times  $l$  asymptotically, we have not achieved the actual goal. Because, we still have a broadcast complexity namely the number of instances of the bit broadcast, that we require which is proportional to the size of the sender's message.

(Refer Slide Time: 21:10)



However, we observe the following here. So, this is the naive way of dividing the sender's message into blocks of size  $(t + 1) \log \log |F|$  bits and running  $B$ s instances of the warm up protocol. We observe that the instances of the bit broadcast is required only in 1 stage in

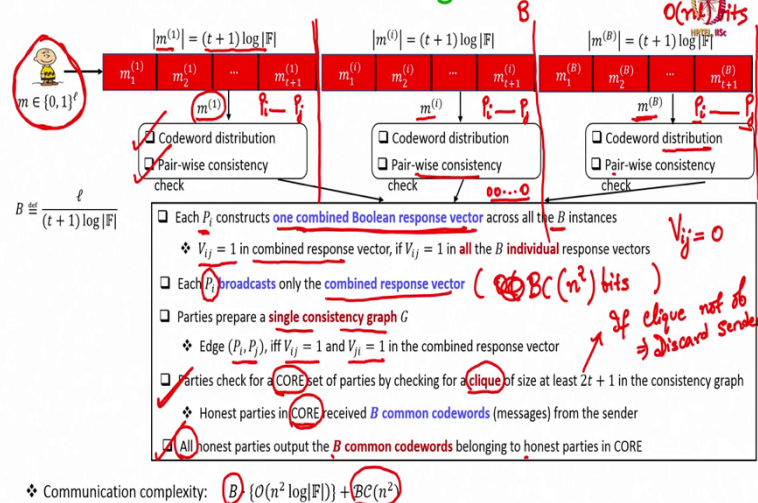
all the  $B$  instances, namely to make public the response vectors. So, there are  $B$  such instances of warm up protocol and that is why there are  $B$  response vectors computed by every party one for one instance of the bit broadcast.

So, in order to reduce the communication complexity or in order to reduce the broadcast cost, namely the number of instances of the bit broadcast. The idea here is that we can club together this highlighted step for all the  $B$  instances of the warm up protocol. Namely, each party will now prepare a single response vector based on the response vector that it has computed for the  $B$  instances.

And, each party will broadcast now a single response vector instead of broadcasting  $B$  response vector. Accordingly, each party will be constructing now a single consistency graph, a single graph for all the  $B$  instances of the warm up protocol. And, based on that a single CORE set will be identified. Let us see how that can be possible.

(Refer Slide Time: 22:57)

## Domain Extension: Reducing the Broadcast Cost



So, this is the way sender has divided its message into big  $B$  number of blocks, where each block consists of  $(t+1) \log \log |F|$  bits. The first two stages of the warm up protocol will be common for all the  $B$  invocations of the warm up protocol. Namely, for the first message block  $m^{(1)}$ , the sender will compute the corresponding code word and it will distribute the corresponding code word. And, then every pair of parties will exchange two supposedly common points on their respective local code words.

In the same way, if we take the  $i$ th block, corresponding to that sender will compute the Reed-Solomon code word, and distribute to the respective parties. And, every pair of parties will exchange the two supposedly common points on their respective local code words. And, like that for the  $B$ th block, a Reed-Solomon code word will be computed. The sender will distribute that code word to respective parties and every pair of parties will exchange two supposedly common points in their respective local code words.

That part will happen for all the  $B$  instances and these two stages will require a communication of order of  $nl$  bits that we have already seen. Now, after this the clubbing happens as follows. Each party  $P_i$  will be now constructing a combined Boolean response vector across all the  $B$  instances of the warm up protocol. So, locally it would have constructed  $B$  number of response vectors ok.

And, in each of the response vectors,  $P_i$  will be either pair wise consistent with the  $j$ th party or it might be in conflict with the  $j$ th party depending upon whether the corresponding supposedly common components between  $P_i$  and  $P_j$ 's local code words are same or not. So, for instance, for the first instance of the warm up protocol, there will be a status between  $P_i$  and  $P_j$  whether they are in conflict or not.

In the same way, for the  $i$ th instance of the warm up protocol the response vector of  $P_i$  will indicate whether  $P_i$  is in conflict with  $P_j$  or not. And, in the same way for the  $B$ th instance of the warm up protocol,  $P_i$  would have prepared a response vector where the  $j$ th entry will indicate whether  $P_i$  is in conflict with  $P_j$  or not. Now, ideally, if the sender is honest and if both  $P_i$  and  $P_j$  are honest, then  $P_i$  should not be in conflict with  $P_j$  in any of these  $B$  instances.

That means  $P_i$  should be consistent with  $P_j$  in the first response vector,  $P_i$  should be consistent with  $P_j$  in the  $i$ th response vector,  $P_i$  should be consistent with  $P_j$  in the  $B$ th response vector. Whereas, if  $P_i$  and  $P_j$  are in conflict in any of the  $B$  response vectors; that means, the sender is corrupt. And, it has not distributed a common code word to  $P_i$  and  $P_j$  in at least one of the  $B$  instances of the warm up protocol.

So, using that intuition what we are now doing is we are asking party  $P_i$  to construct  $B$  response vectors locally, but combine them into a common single response vector which should indicate whether  $P_i$  is in conflict with  $P_j$  in any of the  $B$  warm up protocol instances or not. So, in the combined response vector  $P_i$  and  $P_j$  will not be in conflict, if they are not in conflict in any of the  $B$  instances of the warm up protocol.

But, even if there exists one instance of the warm up protocol, where  $P_i$  is in conflict with  $P_j$ ; then in the combined response vector  $P_i$  will be in conflict with  $P_j$ . So, in which particular instance of the warm up protocol  $P_i$  and  $P_j$  are in conflict, that is not important. So, that is the idea behind constructing this one combined Boolean response vector. And, now even though there are  $B$  instances of the warm up protocol running, each party  $P_i$  will broadcast a single combined response vector to indicate its status with other parties across all the  $B$  instances of the warm up protocol ok.

So, now this stage will require a broadcast of only  $n^2$  bits using any existing bit broadcast protocol ok. Now, based on the combined response vector broadcasted by every party, a single consistency graph is prepared where the condition remains the same. Namely, we will say that parties  $P_i$  and  $P_j$  are fine with each other, if in their respective combined response vectors they are fine with each other. Namely,  $P_i$ 's response against  $P_j$  is 1 and  $P_j$ 's response against  $P_i$  is 1.

And, then parties check whether they exist a CORE set, a single CORE set in this combined consistency graph by checking whether a clique of size at least  $2t + 1$  is there or not ok. If the clique is not obtained in this single consistency graph, then it implies sender is definitely corrupt.

So, discard the sender and take a message consisting of a string of 1 number of 0s on the behalf of the sender. Whereas, if a CORE is obtained; that means, in each of the  $B$  instances of the warm up protocol, the honest parties in the CORE have received a common code word from the sender for that particular instance. Namely, for the first instance the honest parties in the CORE have received a common code word from the sender.

And, like that even in the  $i$ th instance of the warm up protocol, the honest parties in the same CORE have received a common code word from the sender. And, like that even in the  $B$ th instance of the warm up protocol, all the honest parties in the CORE have received the same common code word from the sender, that is guaranteed if a CORE is obtained.

Now, as we have done for one instance of the warm up protocol, what we can do is, in each individual instance of the warm up protocol parties will redefine their respective components of the code words that they would have received in that particular instance of the warm up protocol, based on the communication they have from the parties in CORE.

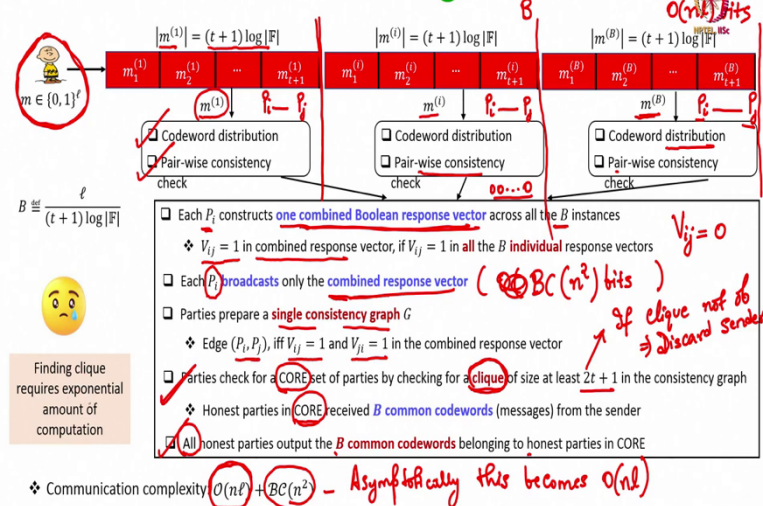
And, then by exchanging that redefined components and using the Reed-Solomon error correction, across all the B instances it will be ensured that the common code word which is held by the honest parties in CORE across these individual B instances of the warm up protocol gets available, is made available to all the honest parties, irrespective of whether they are in CORE or not.

So, with this modification namely combining the response vector across all the instances and then broadcasting a single response vector from by each party, the communication complexity turns out to be the following. The communication over the point to point channel will be B times, the communication complexity for one instance of the warm up protocol.

Namely, for code distribution, pair wise checking and redefining the components of the code words and exchanging them and applying Reed-Solomon code words. So, that stage will have a communication complexity of B times  $n^2 \log \log |F|$  bits. But, the communication which will be done using instances of the bit broadcast will be only  $n^2$  because, each party now broadcast only a single Boolean response vector.

(Refer Slide Time: 32:47)

## Domain Extension: Reducing the Broadcast Cost



And, now if we substitute the value of B, the overall communication turns out to be  $nl$  plus broadcast of  $n^2$  bits using invocations of any existing bit broadcast protocol. And, asymptotically this becomes  $O(nl)$  for a sufficiently large  $l$  ok.

And, that will be the optimal communication you expect from any broadcast protocol because in order that sender's message is available to everyone, sender has to communicate its message to all the  $n$  parties. And, that trivially involves a communication of  $\Theta(nl)$  bits. And, the overall cost of the protocol that we have designed is  $O(nl)$ . However, the downside of this domain extension protocol is that it requires the parties to perform exponential amount of computation.

Because, to check whether a CORE is present in the system or not, we have to check whether a clique of size at least  $2t + 1$  is present in the consistency graph and finding clique requires performing exponential amount of computation. So, even though communication wise we have now got a very good communication efficient bit broadcast protocol, the amount of computation which is required in the protocol is exponential time. So, now our next goal will be to somehow modify the protocol; so, that the protocol does not require performing exponential amount of computation.

(Refer Slide Time: 34:42)

## References



- ❖ Arpita Patra: Error-free Multi-valued Broadcast and Byzantine Agreement with Optimal Communication Complexity. OPODIS 2011: 34-49

So, with that I end this lecture. The reference used for today's discussion is this paper

Thank you.