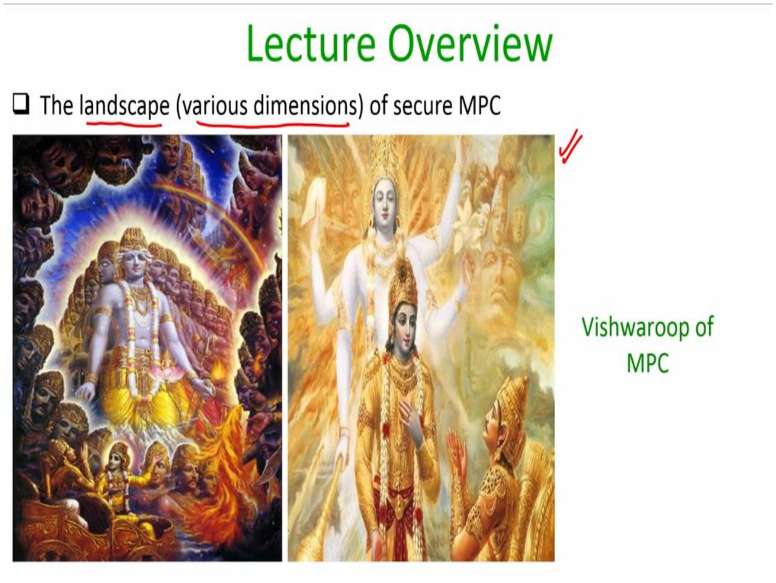
Secure Computation - Part I
Prof. Ashish Choudhury
Department of Computer Science
International Institute of Information Technology, Bangalore

Module - 1
Lecture - 3
Various Dimensions to Study Secure MPC

(Refer Slide Time: 00:33)



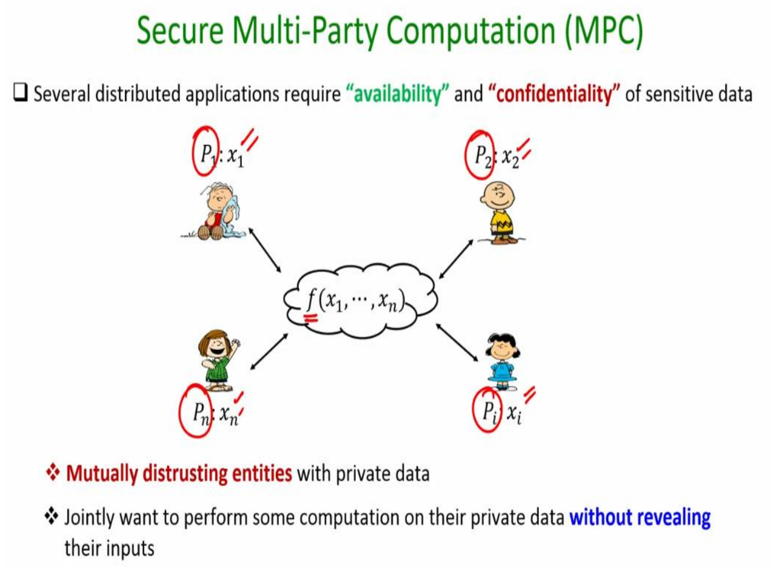
Lecture Overview

- The landscape (various dimensions) of secure MPC

Hello everyone. Welcome to this lecture. So, the plan for this lecture is as follows: We have seen on a very high level, what is the secure multi-party computation or MPC problem. We had seen some real-world examples of secure multi-party computation. It turns out that the MPC problem can be studied in varieties of domains. It is enormously large. And in this lecture, we will see the landscape, namely the Vishwaroop or the various dimensions of secure MPC.

So, people who are not familiar with this picture, in the Battle of Kurukshet, Lord Krishna showed his enormous landscape or various dimensions, which we call as Vishwaroop. In the same way, the goal of this lecture is basically to show you the various dimensions in which we can study the secure MPC problem.

(Refer Slide Time: 01:32)



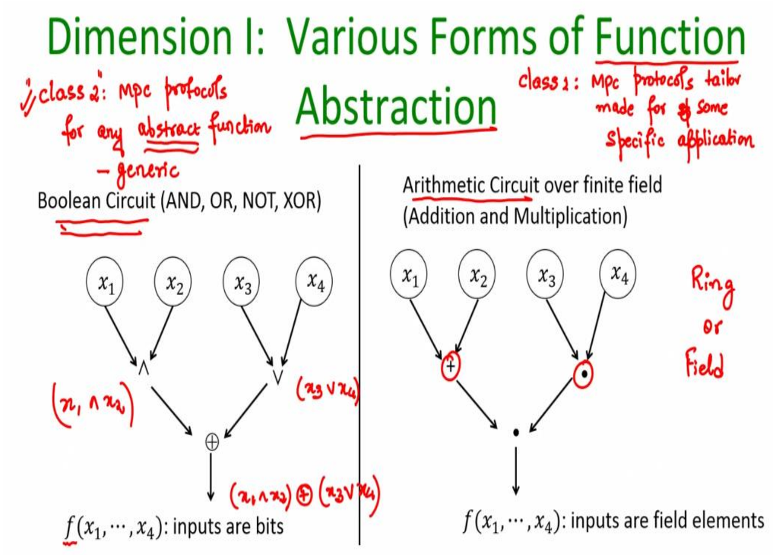
Secure Multi-Party Computation (MPC)

- Several distributed applications require **"availability"** and **"confidentiality"** of sensitive data

- **Mutually distrusting entities** with private data
- Jointly want to perform some computation on their private data **without revealing** their inputs

So, just to recall, this is the blueprint of secure multi-party computation. We have set of mutually distrusting parties with some private data. And there is some publicly known function f. And the goal is to design a mechanism which allows the parties to obtain the result of this function f on the inputs $x_1, \ldots, x_n$, and in the process learn nothing about other party's data other than what a party can itself infer from its own input and the function output.

(Refer Slide Time: 02:15)



Dimension I: Various Forms of Function Abstraction

i) class 2: MPC protocols for any abstract function – generic

class 1: MPC protocols tailor made for ~~a~~ some specific application

Boolean Circuit (AND, OR, NOT, XOR)

Arithmetic Circuit over finite field (Addition and Multiplication)

x_1
x_2
x_3
x_4
$\wedge$
$\vee$
$(x_1 \wedge x_2)$
$(x_3 \vee x_4)$
$\oplus$
$(x_1 \wedge x_2) \oplus (x_3 \vee x_4)$
$f(x_1, \cdots, x_4)$: inputs are bits
$+$
$\cdot$
$f(x_1, \cdots, x_4)$: inputs are field elements
Ring or Field

So, the first dimension is based on what are the various forms of abstraction that you can follow to represent the function to be securely computed. So, what do I mean by that? It turns out that we have 2 classes of MPC protocols. Class 1 constitutes of MPC protocols tailor made for some specific application. Say for instance, your application is E-auction, where the function is to compute the max of n bids.

So, I might design an MPC protocol which just helps me to solve that problem. That protocol may not help you to find out or solve the problem of privacy-preserving machine-learning. So, these protocol, which are tailor made for some specific application, are called class 1 MPC protocols.. You give me the application and I design the protocol only for that specific computation. Class 2 is the class of MPC protocols for any abstract function.

That means, it is not tailor made only for secure auction; it is not tailor made only for millionaires problem; it is not tailor made only for machine learning application and so on. It is for any function f that you give me. That means that the MPC protocol will work irrespective of what exactly is your function f. But, from the description, it might look like that the class 2 function, the class 2 MPC protocols are more generic.

So, basically they are generic, because that can be deployed for any specific instantiation of your function f. But when I am designing a generic MPC protocol, I have to follow some representation for the function f. Because, if I do not know any details about the function f, how can I design a protocol? And how can I design an algorithm to securely compute that function?

So, I have to do some abstraction or I have to represent my function f, underlying function f in an abstract fashion, and that is what I mean by function abstraction. So, there are 2 popular forms of function abstraction which are used in class 2 MPC protocols or any generic MPC protocol. The first form of function abstraction is Boolean circuit representation, where we assume that the function f which the parties want to securely compute is represented by some Boolean circuit.

And the description of the Boolean circuit will be publicly known to all the n parties. And it is a well-known fact that you take any computation, any function f; it can be always represented by some Boolean circuit consisting of AND gates, OR gates, NOT gates, various other kinds of gates. So, for instance, this is an example of a Boolean circuit over 4 input bids $x_1, x_2 x_2$ and x_4. And the function that is represented by this circuit is the following:

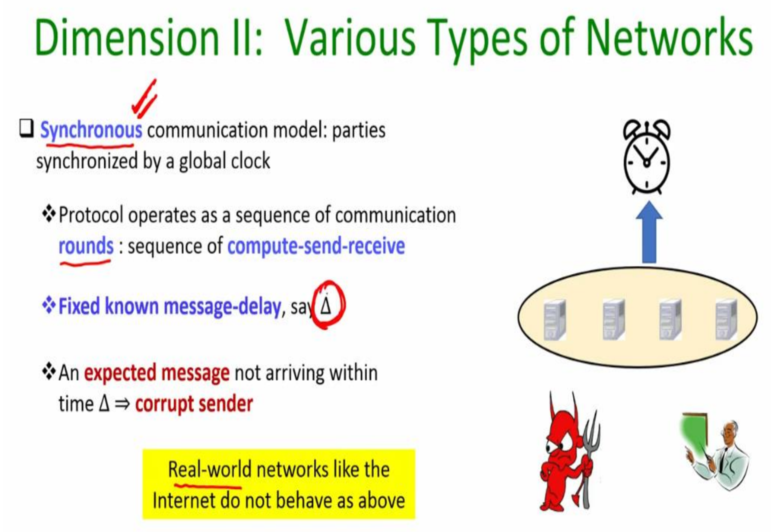
So, this is your AND gate. So, you want to compute x_1 and x_2. This is an OR gate, you want to do x_3 or x_4. And then finally, you want to compute XOR. So, this could be an example of an abstract function. So, in a Boolean circuit abstraction, we assume that the function f which the parties want to securely compute is represented by some Boolean circuit consisting of Boolean gates, and the details or the topology of that circuit is publicly known to all the n parties.

Whereas, the second abstraction used in class 2 MPC protocols is that of arithmetic circuit representation, where we assume that the function f which the parties want to securely compute is represented as some circuit over some algebraic structure which could be either a ring or a field. So, we will see soon what exactly a ring, a field is. So, the point here is that, in this abstraction, we assume that whatever computation the parties want to perform, it is represented by some publicly known circuit.

And the circuit consists of addition gates and multiplication gates over the ring or a field. And all the operations here are performed over the ring or the field. Now, we have various generic MPC protocols designed for Boolean circuit. We have MPC protocols designed for arithmetic circuit. And depending upon our convenience, upon our requirement, various efficiency issues, and other aspects, we can decide whether to opt for an MPC protocol to securely compute a Boolean circuit or whether to go for an MPC protocol to securely compute or securely evaluate an arithmetic circuit.

So, that is the first dimension for studying MPC protocols. So, you have MPC protocols for Boolean circuit functions; you have MPC protocols for arithmetic circuit functions.

(Refer Slide Time: 09:22)



Dimension II: Various Types of Networks

- Synchronous communication model: parties synchronized by a global clock
    - Protocol operates as a sequence of communication rounds : sequence of compute-send-receive
    - Fixed known message-delay, say Δ
    - An expected message not arriving within time $\Delta \Rightarrow$ corrupt sender

Real-world networks like the Internet do not behave as above

Let us see the second dimension, depending upon what kind of underlying network the parties have. So, in this dimension, we can either have synchronous communication model or asynchronous communication model. Let us first try to understand the synchronous communication model. In the synchronous communication model, the parties are synchronized by a global clock.

And this means that the parties will be talking to each other in the MPC protocol, and the protocol will operate as a sequence of rounds. What do we mean by rounds? In each round, as part of the MPC protocol, a party will perform some computation, based on what input it has. And then it will send the messages that it has computed to the other parties. And what the other parties have computed in that specific round will be received by the parties.

That will finish one round. And then they will perform this cycle for the next round. In the next round, whatever messages I have received from the other parties at the end of the previous round, I will process them as per the MPC protocol instructions, compute the next set of messages I have to send to the other parties, and I will send them. That will finish the second round. And then the third round will start.

So, if my protocol is an r round protocol where r could be 1, 2, 3; in each round, each party will perform this operation. And there will be a fixed delay for the messages. That means, in each round, a party will know that, okay, the other parties are expected to send some messages; those messages should arrive to me by the end of this clock cycle. Because, we are assuming that we are in a synchronous communication model and the clocks of all the n parties, they are synchronised.

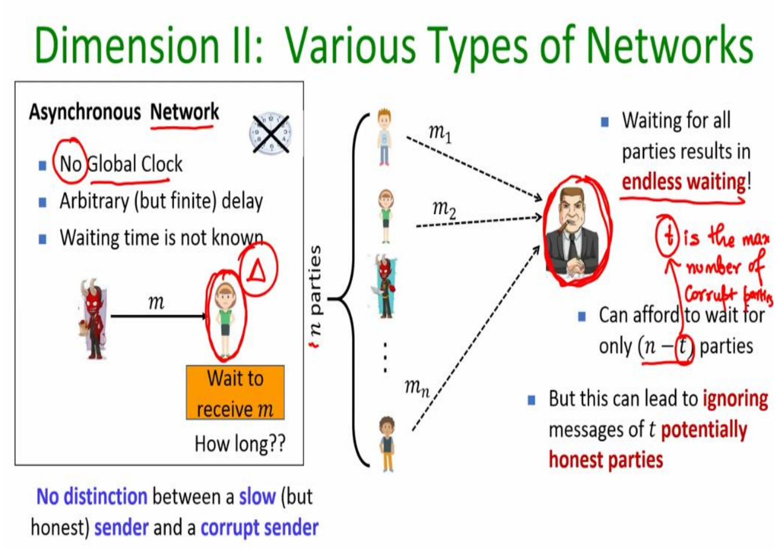
So, if the delay is, say 1 hour; that means, each round operates for 1 hour; then I know that each party is supposed to send its messages to me within 1 hour. And then, by the end of the 1 hour, the next round will start. Whatever I have to send to the other parties, I will compute and send it. And the other parties are also supposed to do the same and send their messages which should come to me by the end of the 1 hour.

That means, the message delay will be publicly known. And since the message delay is publicly known, that means, if the message delay, say for instance, is 1 hour, and within 1 hour, an expected message which I am expecting from one of the parties as part of the protocol does not come to me, then I can conclude that, okay, that sending party is a corrupt party.

And then, as part of the MPC protocol, I can decide how to deal with that corrupt party for the rest of the protocol instructions. So, an implication of the synchronous communication model is that, if a sending party does not send its expected messages, then the receiving party can conclude that the sender is corrupt. However, even though synchronous communication model is very simple abstraction, your real-world networks like the internet do not behave like a synchronous communication model.

Because, if one of the parties is in one end of the globe, and the other party is in the other end of the globe, and they are connected over the internet, then we cannot come up with the exact value of delta. It might be the case that the network is very slow, and we do not know how long it takes for a message from party 1 to receive to the party 2. The network might be very slow, sometimes the network might be very fast, and so, we cannot come up with a precise bound for delta.

(Refer Slide Time: 13:52)



Dimension II: Various Types of Networks

Asynchronous Network

- No Global Clock
- Arbitrary (but finite) delay
- Waiting time is not known

No distinction between a slow (but honest) sender and a corrupt sender

- Waiting for all parties results in endless waiting!
- Can afford to wait for only $(n - t)$ parties

t is the max number of corrupt parties

- But this can lead to ignoring messages of t potentially honest parties

To model this fact, we consider another form of communication model which we call as asynchronous communication model. And in the asynchronous communication model, there is absolutely no global clock. The parties are not at all synchronised by any kind of clock. That is why you cannot define the notion of a round. Because, since the message delays are not known, and if I am a receiving party, and if I am expecting a message from one of the parties as part of the MPC protocol, and if that message is not coming to me, I do not know how long I have to wait for the message.

It is like saying the following: Only when I receive a phone call from one of the parties in the MPC protocol and receive that message over the telephonic conversation, I can proceed to my next step in the protocol. But I do not know when the phone will come, when the call will come. It could be 1 day; it could be 1 hour. And I cannot come to a conclusion that, okay, since I have waited enormously long and the call has not come, the sending party or the party who was supposed to make that call is a corrupt party.

I cannot do that, because it might be the case that the next instance itself, the next moment itself call comes, as soon as I decide that, okay, that party is a corrupt party. So, the major challenge in an asynchronous communication model is that, from the viewpoint of a receiving party, the receiving party cannot make a distinction between a slow party or a slow sender and a corrupt sender.

Namely, from the viewpoint of this receiver, if it is expecting for a message m and if the message is not coming or has not yet come, then this receiver cannot distinguish apart whether the sender was corrupt and purposely has not sent a message or the sender was honest, it has sent the message, but the message got arbitrarily delayed. So, from the viewpoint of the receiver, both these situations can occur, unlike a synchronous communication model.

Because, in the synchronous communication model, the receiver will know that, okay, I have to wait only for time delta. And if within the time delta the message does not come, definitely the sender is corrupt. Because, in a synchronous communication model, each expected message is supposed to be delivered within this delta time frame. So, if we are considering an n party MPC protocol or n party protocol for any distributed computing task in an asynchronous communication model, then no receiving party can afford to receive communication from all the n parties.

Because, there might be some corrupt parties among these n parties, who may not send their expected messages. And if this receiving party is waiting for all the parties to send their messages, then it might lead to an endless waiting. And the protocol may not terminate at the first place. So, to avoid this scenario, what we do is the following, in the asynchronous communication model:

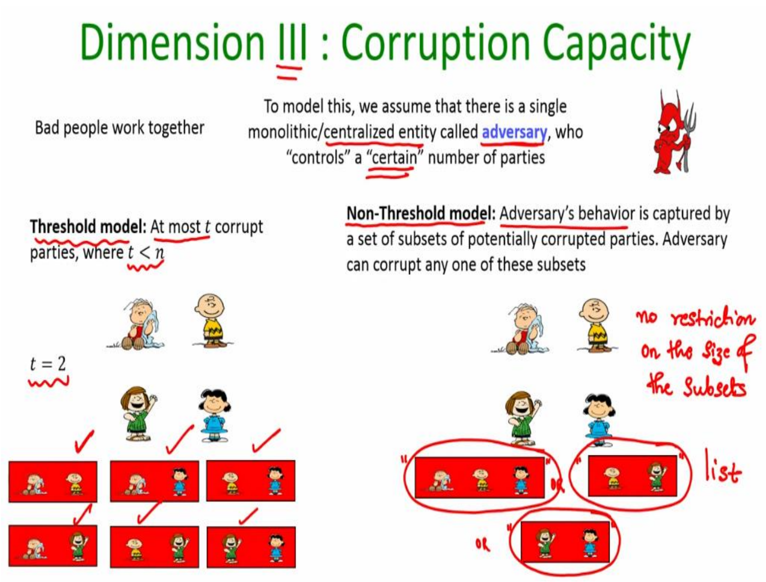
If we assume that t is the maximum number of corrupt parties in the system, then in an asynchronous protocol, a receiving party can afford to receive communication from at most $n - t$ parties. It cannot afford to receive communication from the remaining t parties as well. Because, the remaining t parties might be corrupt parties, because t is assumed to be the upper bound on the number of bad parties.

And bad parties may decide not to send their messages. So, in asynchronous protocol, as soon as a party receives communication from $n - t$ parties, that step is done, and it has to go to the next step. That means, at each step, communication from t parties have to be ignored. But the problem here is that, it is not the case that the t parties from whom this receiving party have ignored the communication are always the bad parties.

It could be the case that those t parties whose communication or messages are ignored are actually slow parties. That means, they have sent the messages, but they got stuck in the network, and that is why it got delayed. But this receiving party, it does not know that, okay, those sending parties are honest or corrupt, and whether those messages are going to come or not. It has simply decided that I am going to ignore those t messages and I have to proceed to the next step of the protocol.

So, that is why, this phenomena will always be encountered in an asynchronous protocol. And that is why, compared to the synchronous protocols, asynchronous protocols are more complex, but at the same time, they are more practical. Because, at the end of the day, when you are going to implement your MPC protocol, it has to be implemented over a real network. And real-world networks like the internet are asynchronous in nature.

(Refer Slide Time: 19:41)



Dimension III : Corruption Capacity

Bad people work together

To model this, we assume that there is a single monolithic/centralized entity called adversary, who "controls" a "certain" number of parties

Threshold model: At most t corrupt parties, where $t < n$

Non-Threshold model: Adversary's behavior is captured by a set of subsets of potentially corrupted parties. Adversary can corrupt any one of these subsets

$t = 2$

no restriction on the size of the subsets

or

or

list

Now, let us see that I mentioned number 3, according to which we can categorize our MPC protocols. And this dimension is based on the corruption capability. So, remember, I am continuously saying that we have a set of mutually distrusting parties. They do not trust each other. So that people always work together. And among the n parties, we have 2 categories of parties. One category of parties who are "good".

They are just interested to get the result of the computation, nothing else. And there is another category of parties, which you could consider as a "bad" set of parties, which would like to cause harm to the good parties, either in terms of learning their inputs or causing them to obtain the incorrect result of the computation and so on. So, to model the bad people in the system or bad people among the n parties, we assume that there is a centralized abstract entity called adversary who "controls" a "certain" number of parties among those n parties, in varieties of ways.

So, that is why I have written controls and certain in quote-unquote. Depending upon how many parties the said adversary can control, we have either a threshold adversary model or a non-threshold adversary model. So, in the threshold model, which is the more popular adversary model, we design MPC protocols assuming that during the protocol execution, at most t parties could be corrupt.

By corrupt means, they can fall under the control of the adversary, and of course, that t has to be lesser than n. Because, if all the n parties are corrupt, then how or why at the first place you are interested to preserve the privacy of the individual data. So, we assume that in the threshold model, any set of t parties out of the n parties could get corrupt during the protocol execution.

However, which t parties are going to be corrupt during the protocol execution, we may not be knowing that. And in fact, at the end of the protocol execution also, we may not learn that. So, our goal is to design a mechanism which should ensure the parties to securely do the computation even if up to t parties get corrupt during the protocol execution, and it could be *any* set of t parties out of the n parties.

So, for instance, if I say $t = 2$, then, during the protocol execution, it could be any set of t parties which could be corrupt, which could fall under the control of the adversary when the protocol gets executed. That is a threshold adversary model. A more generalized form of the adversary is the non-threshold model where we do not bound the corruption capacity by a number.

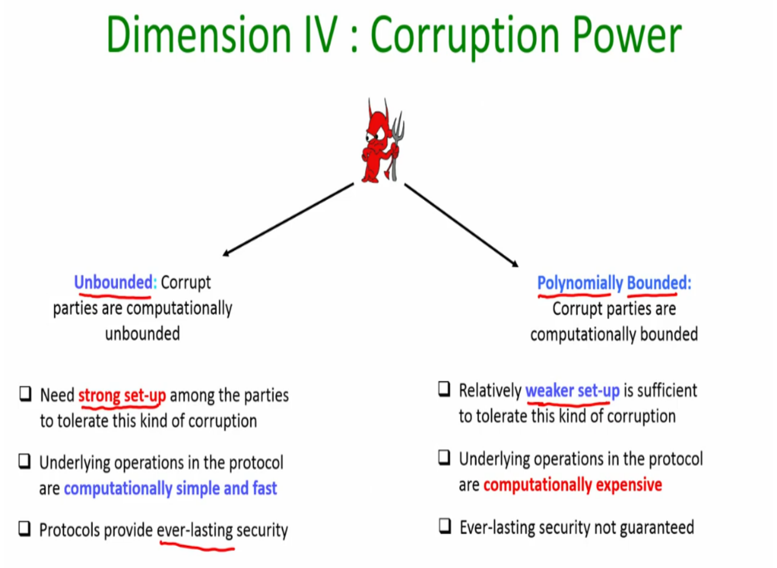
Rather, here we do the following: We see that the adversary's behavior, namely, its corruption behavior is specified by mentioning or listing down the subsets of potentially corrupt parties. And during the protocol execution, any one of those sets which I have listed down, could get corrupt by the adversary. So, again, let us take the example where we have 4 parties P_1, P_2, P_3 and P_4. And I can specify a list like this - $\{\{P_1, P_2, P_3\}, \{P_2, P_4\}, \{P_1, P_3\}\}$.

I can say that, okay, design a MPC protocol which should work; work in the sense, which should remain secure; even if this set of 3 parties gets corrupt, or this set of 2 parties gets corrupt, or this set of 2 parties gets corrupt. That means, your goal will be now to basically handle either this collection; handle in the sense, to take care of the fact that even if this set fails; fails means get corrupt and fall under the control of the adversary; still your protocol should work.

Or it should work even if this subset falls or if this subset falls. Now, why this is more generic? It is more generic because, in this list, I am not specifying, I am not putting any restriction on the size or the cardinality of this individual subsets. So, no restriction on the size of the subsets. Different subsets could be of different size. This is unlike your threshold communication

model, where in the threshold communication model, it is only subsets of size t which could get corrupt and fall under the control of the adversary.
It cannot be that; it could be a subset of $t = 2$ or subset of $t = 3$. Once you fix the parameter t, only those number of parties could get corrupt. But in the non-threshold model, it could be different subsets of different size which could fall under the control of the adversary. So, in some sense, you can imagine that threshold adversary model is a special form of non-threshold model, where the size of each subset of potentially corrupt parties is upper bounded by t.

So, these are the 2 ways in which we can specify the corruption capacity of the adversary. And we have MPC protocols for threshold model; we have MPC protocols for non-threshold model.
(Refer Slide Time: 25:58)



Dimension IV : Corruption Power

Unbounded: Corrupt parties are computationally unbounded

- Need **strong set-up** among the parties to tolerate this kind of corruption
- Underlying operations in the protocol are **computationally simple and fast**
- Protocols provide ever-lasting security

Polynomially Bounded: Corrupt parties are computationally bounded

- Relatively **weaker set-up** is sufficient to tolerate this kind of corruption
- Underlying operations in the protocol are **computationally expensive**
- Ever-lasting security not guaranteed

Dimension 4 is based on the corruption power of the adversary. So, remember adversary is at the end of the day, some algorithm run by some computer. Now, the adversary could be computationally unbounded. That means, we do not put any restriction on the resources, specifically the computing resources of the adversary. It is allowed to do any kind of computation.

It can do brute force; it can perform exponential computation; we do not put any restriction whatsoever on the computing speed or the computing power of the corrupt guys. Whereas, we have another category of adversaries where we say that adversary is polynomially bounded. That means, the corrupt parties are only allowed to perform limited amount of computation. Now, there are both pros and cons of MPC protocols, tolerating an unbounded adversary and a bounded adversary.

If you want to design MPC protocols tolerating an unbounded adversary, then we need to have very strong setup among the parties. So, we will see later, what I mean by a strong setup. Loosely speaking, we require that the parties should be able to communicate to the other parties in a secure way even in the presence of computationally unbounded adversaries. Whereas, if we want to design protocols tolerating computationally bounded adversaries, then we do not require such strong setup; even a weaker setup is sufficient to tolerate this kind of corruption.

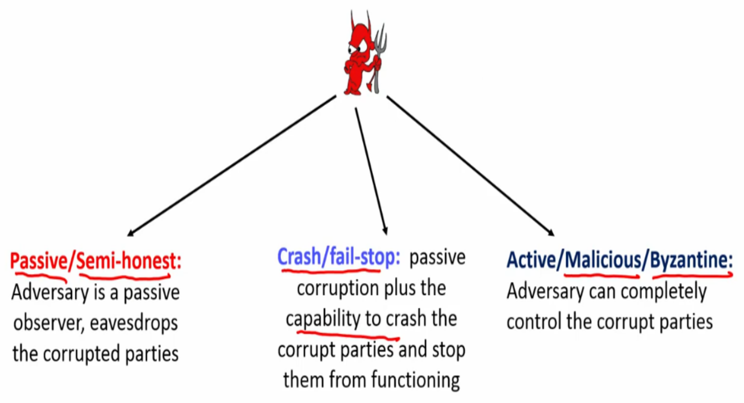
However, the protocols which are designed to tolerate computationally unbounded adversaries, they are very simple and very fast in terms of the computations or steps which are there in the protocol. That means, those computation, those steps can be very easily performed by the parties, and it will take very less time compared to the protocols tolerating computationally bounded adversaries, where the computation steps are computationally very expensive.

Also, the protocols, MPC protocols tolerating computationally unbounded adversaries provide you everlasting security. Why it will provide you everlasting security? Because, those protocols give you security guarantees, even if the adversary is computationally unbounded. That means, even if, say, quantum computers become reality, the security offered by the protocols will be preserved, compared to the protocols which gives you security only against polynomially bounded adversaries.

Because, in this latter class of MPC protocols, the security will be achieved as long as some standard cryptographic assumptions hold. But as soon as those cryptographic assumptions are broken, the security guarantees provided by those MPC protocols are also broken. So, that is why this latter class of MPC protocol does not provide you everlasting security. So, that is your dimension number 4.

(Refer Slide Time: 29:41)

Dimension V : Corruption Characteristic


Passive/Semi-honest: Adversary is a passive observer, eavesdrops the corrupted parties
Crash/fail-stop: passive corruption plus the capability to crash the corrupt parties and stop them from functioning
Active/Malicious/Byzantine: Adversary can completely control the corrupt parties

Dimension number 5 classifies MPC protocols depending upon the corruption characteristic of the adversary. So, remember, adversary can control certain number of parties. What do I mean by control? Depending upon the control mechanism, we have passive corruption, which is also sometimes called as semi-honest corruption or eavesdropping corruption. So, here, the goal is the following:

The adversary basically is a passive eavesdropper and it can only eavesdrop the corrupt parties. What do I mean by that? So, in an MPC protocol, the parties will exchange messages, and at the end of these message exchanges, they will obtain the result of the computation. Now, the parties which are not under the control of the adversaries, they would not be analyzing what messages they are receiving.

Analyzing in the sense, they will analyze it as per the MPC protocol, they will obtain the result of the computation, and that is all, the task is done for them. But an eavesdropper might do the following: Once the result of the computation is achieved, it now might try to analyze those messages. That is not done as part of the MPC protocol. It might deploy its own resources to analyze those messages and try to learn something about the inputs of the other parties or the honest parties.

By honest parties, I mean the parties which are not under the control of the adversary. So, in the eavesdropping model or in the passive corruption model, we assume that the bad guys who are under a control of the adversary honestly follow the protocol instructions. That means, what are the steps specified for them as per the MPC protocol, those steps will be executed as it is.

But also, on top of that, the adversary might try to analyze the messages received by those parties and try to learn something additional about the inputs of the other parties. So, it is like saying the following: You have n computers and the computers are running the MPC code, and a virus has been injected in some of the machines. And all those viruses are monitored by a single adversary.

And through those viruses, whatever messages those specific computers which are under the control of the virus are received, those messages will be transferred to the adversary, and adversary might be analyzing those messages. But that virus would not prevent the corrupt machines to deviate arbitrarily. The instructions run, the program run by those machines, they will run as they should.

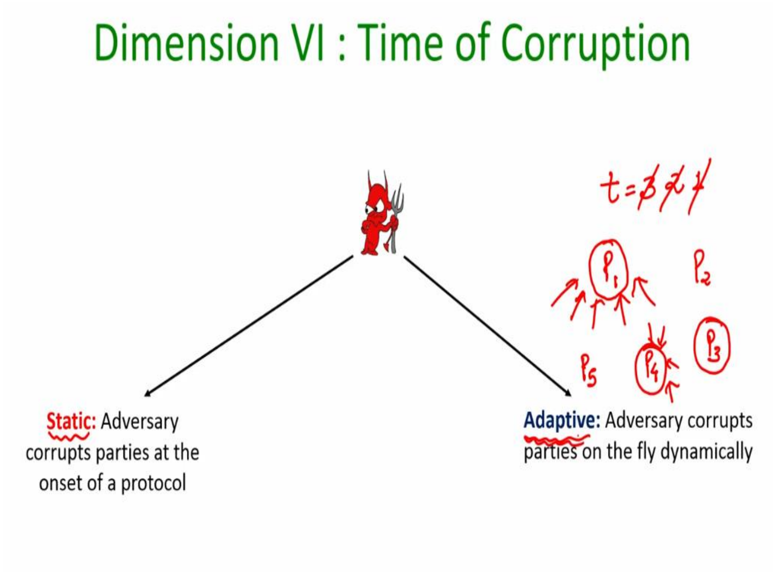
So, that is what we mean by passive or semi-honest corruption. A more powerful adversary model is the crash model or the fail-stop model, where the adversary, apart from eavesdropping, also has the capability to crash the corrupt parties. And once a party crashes during the protocol execution, it stops functioning from that point onward. So, it is like saying the following:

The adversary, it has 2 options; either it can let the corrupt parties finish the protocol execution as allowed in the passive adversary model. But on top of that, the adversary also has the flexibility that any of the machines which are under the control of the adversary stop functioning. And once it stops functioning, it will not come alive again till the end of the protocol. So, that is a crash model, fail-stop model.

And the most powerful corruption characteristic is called as the Byzantine corruption, also known as active corruption or malicious corruption. And here, the corrupt parties, the parties which are under the control of the adversary, can start behaving in any arbitrary fashion. That means, it is up to the adversary. If it wants, it can let the corrupt machines honestly follow the protocol instructions; that means, it can let them behave as if they are behaving in a semi-honest fashion; or it can crash them at any time; or it can instruct them to start sending wrong messages.

That means, the complete control is now given to the adversary. It can dictate the behavior of the corrupt machines in whatever way the adversary would like to. So, that is a malicious corruption or active corruption, also known as Byzantine corruption.

(Refer Slide Time: 34:40)



Dimension VI : Time of Corruption

Static: Adversary corrupts parties at the onset of a protocol

Adaptive: Adversary corrupts parties on the fly dynamically

The sixth dimension classifies your MPC protocols depending upon the time of corruption. And in this category, we have 2 classes of MPC protocols. We have protocols tolerating a static adversary, where the nature of the adversary is that it has to decide the set of corrupt parties at the onset of the protocol. That means, even before the protocol execution starts, the adversary has to decide that, okay, these are the machines, these are the parties whom I am going to control.

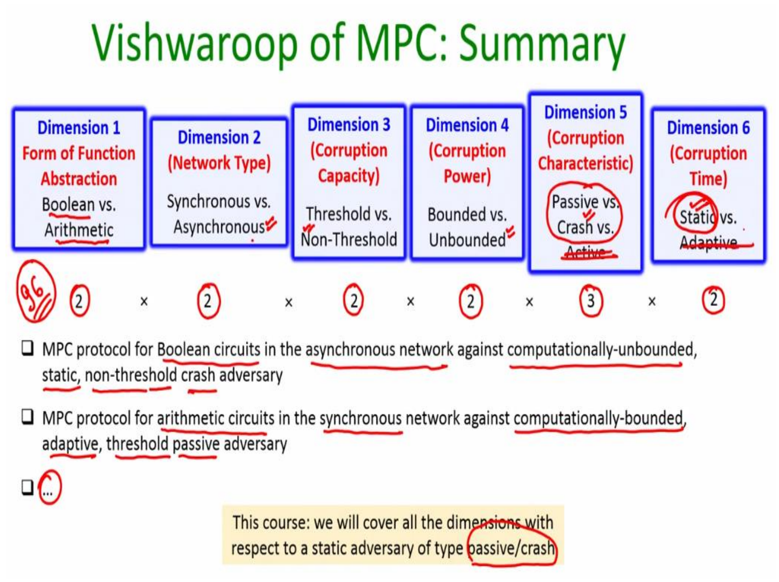
Whereas, a more powerful adversary model is the adaptive corruption model, where adversary has the flexibility to corrupt parties on the fly, during the runtime of the protocol. So, for instance, if I am considering threshold adversary with adaptive nature of corruption and say $t = 3$ and say there are 5 parties who are participating in the protocol, then it might be the case that, to begin with, adversary has decided not to corrupt any parties.

Now, say my protocol is a protocol which operates for 100 rounds, there are 100 rounds of communication, assuming that I am designing a protocol in the synchronous model. Now, an adaptive adversary can do the following: It can decide that, okay, at the end of second round, I am going to corrupt P_1. That will reduce t from 3 to 2. Now, depending upon whatever messages P_1 has received, adversary may decide that, okay, now I am going to corrupt P_4.

That will make t from 2 to 1. That means, now, adversary will have control over the messages sent and received by both P_1 and P_4. And now, adversary might analyse and decide that, okay, I am now going to corrupt P 3. So, that is a form of adaptive corruption. An adaptive adversary could also do the following: It can behave like a static adversary as well. It can say, okay, I am going to corrupt P_1, P_3, P_4 at the onset of the protocol itself.

The point is that, we are now giving the adversary more flexibility to attack your MPC protocol. And clearly, your adaptive corruption model is more powerful than your static corruption model.

(Refer Slide Time: 37:31)



Vishwaroop of MPC: Summary

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

$96 \quad 2 \times 2 \times 2 \times 2 \times 3 \times 2$

- MPC protocol for Boolean circuits in the asynchronous network against computationally-unbounded, static, non-threshold crash adversary
- MPC protocol for arithmetic circuits in the synchronous network against computationally-bounded, adaptive, threshold passive adversary
- …

This course: we will cover all the dimensions with respect to a static adversary of type passive/crash

So, here is the summary of the various dimensions in which we can study the MPC problem. Depending upon what form of function abstraction we use, we can have MPC protocols either for securely computing Boolean circuits or for securely computing arithmetic circuits. So, we have 2 class of MPC protocols here. Depending upon the network type, you can either have MPC protocols in the synchronous communication model or MPC protocols in the asynchronous communication model.

Depending upon whether the corruption capacity is threshold or non-threshold, you have 2 categories of MPC protocols. Depending upon whether the computing resources of the adversary is bounded or unbounded, you have 2 categories of MPC protocols. Depending upon whether the corruption characteristic of the adversary is passive or fail-stop or Byzantine, you have 3 categories of MPC protocols.

And finally, depending upon at what time the adversary strikes, you have 2 categories of MPC protocols. And each of these dimensions are independent of each other. So, that is why, if you take the Cartesian product of various possibilities under various dimensions, that will roughly give you the total number of ways in which you can study your MPC protocols. You have 2 into 2, 4, 8; 8 to 16, 48, 96 different dimensions in which you can study the MPC problem.

So, for instance, I might design a protocol for securely computing Boolean circuits; that means, I have taken from this dimension 1, Boolean circuits. For the asynchronous communication model; that means, from dimension 2, I take asynchronous model. Tolerating a computationally unbounded adversary; that means, from dimension 4, I have taken unbounded adversary who is static and it can control the corrupt parties as per a list of potential subset of parties which is given, namely non-threshold model. And where the corruption characteristic is crash; so, that means, what I am saying here is, if you take under each dimension 1 possibility and take them together, that gives you 1 possible dimension in which you can design MPC protocol or 1 type of MPC protocol. And as I said, there are 96 possibilities.

In the same way, you can design MPC protocol for securely evaluating arithmetic circuits, but your network is now synchronous and your adversary could be computationally bounded, adaptive in nature, who can control up to t parties, but in a semi-honest way. And as I said that this list is exhaustive. And it turns out that, for each dimension, the MPC protocol that you design is different.

It is not the case that a protocol that you designed in 1 dimension will always work in the other dimension. Of course, if you have a weaker dimension compared to a stronger dimension, then the protocol in the stronger dimension will work for the weaker dimension as well. But the other way around may not be possible. But you might design an efficient protocol tailor made for the weaker dimension compared to the protocol designed for the stronger dimension.

So, it is not always that, okay, you design a protocol for the strongest possible adversary model and use it for all the weaker possible adversary model. That is actually a bad design principle. Depending upon what specific dimension I am considering, I would be interested to design a protocol which is the best possible protocol for that specific dimension. And, as you can see that there are so many dimensions possible.

It is not at all possible to cover all these dimensions in just a single course. And by the way, I have just considered 6 possible ways to categorize your MPC protocols. There are of course more sophisticated and advanced ways to categorize your MPC protocols, which are not listed down in this presentation. So, in this course, we will cover all the dimensions, but with respect to a static adversary and where our adversary could be either of type passive or fail-stop.

We will not be considering active adversary model and adaptive corruption in this course, because that is very advanced. And it turns out that even for studying various MPC protocols, state of the art for static corruption and type passive and fail-stop, a 12 week course content will be very short, very less. So, that is why, my intention earlier was to cover all the dimension in a single course.

But when I started designing this course, I realized that the techniques are so sophisticated, so advanced, that I would not be doing a proper justice if I cover all the dimensions in a single course. And that is why, in this specific course, we will stick only to the passive and crash model. We will cover Boolean circuit, arithmetic circuit, synchronous, asynchronous, threshold, non-threshold model.

We will see both computationally bounded adversaries, unbounded adversaries. But we will not be considering active corruption and adaptive mode. Those protocols techniques will be covered in the second part of this course. With that, I end my lecture for today. Thank you.