CSA E0 312: Secure Computation	August 21, 2015
[Lectures 2-4]	
Instructor: Arpita Patra	*Submitted by: Ajith S*

1 Expanding the Scope of Multi-party Computation (MPC)[i]

In this lecture we will see how vast, broad and rich the area of MPC is. We will see its expanse in details expanding over multitude of facets and dimensions that are orthogonal to each other. It is a primitive that is strong enough to capture all most everything, if not everything in cryptography.

1.1 Dimension 1: Models of Computation

So far we have seen how to securely compute addition over finite fields and multiplication of bits. They are just specific examples of MPC. MPC in fact allows to securely compute any polynomially computable (PPT) function. How do we compute any PPT function securely using MPC? This is where we appeal to complexity theory and a given function is first transformed to a finite size circuit. Two circuit models that are suggested are (Figure 1)

– Boolean Circuit

– Arithmetic Circuit over a finite field

A **Boolean** circuit consists of gates such as AND, OR, XOR, NOT etc and takes bits as inputs. On the other hand, an **arithmetic** circuit consists of addition and multiplication (over the finite field under consideration) gates. Once we have the circuit representation of a given function, MPC works by evaluating the circuit in a 'secure' way. Meaning that, the circuit is evaluated in a way that ensures none of the values assigned to circuit input wires and intermediate gate output wires will be leaked to the participating parties. Only the value for the circuit output wire that represents the function output will be revealed to all. Recall that the aim of MPC is to allow the parties to compute the function output and nothing beyond.

A question that may arise now is: which of these two circuit models should we prefer? It depends on the function that we want to compute. For a function f, one will prefer the model in which the representation of f is concise i.e. the circuit size (the no. of gates in the circuit defines the circuit size) is smaller. As we will see later, the efficiency of MPC is directly proportional to the circuit size. To be specific on the choice of the circuit model, let us consider the following example. Consider the function $f(x_1, x_2) = x_1 + x_2$. In Boolean circuit, addition of two n bits number will require at least n gates. On the other side, the arithmetic circuit over an appropriate field will have only one addition gate. As a general practice, non-linear operations such as comparison, greater than, less than etc are represented in Boolean circuits, while operations on polynomials are represented using

[i]This section was covered in Lecture 2 on August 12, 2015

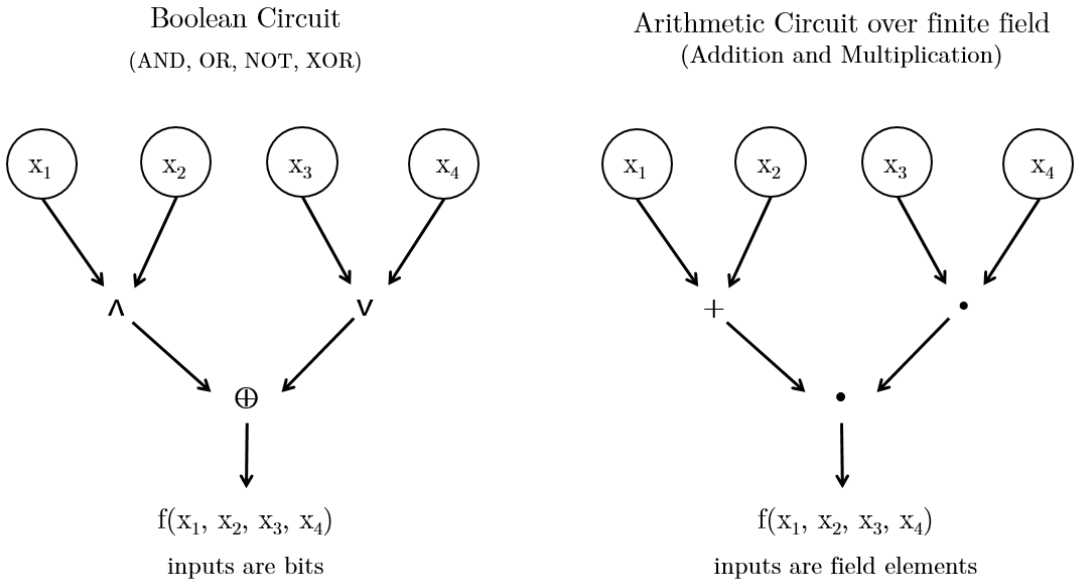

Figure 1: Models of Computation (Boolean Circuit vs. Arithmetic Circuit)

Arithmetic circuits. Some works on MPC try to optimize the circuit size of a given function by decomposing the function into several sub-functions and finding representation for the sub-functions in arithmetic circuit or Boolean circuit model whichever gives the optimal representation. We will refer to such circuits that are partly Boolean and partly arithmetic as *hybrid circuit*.

While both arithmetic circuits over finite fields and Boolean circuits are well-studied model of computation, the hybrid circuit model is very less explored and thus leaves a huge scope of work.

1.2 Dimension 2: Varieties of Network

The MPC protocols are interactive in nature. The parties participating in MPC must communicate with each other to compute a function securely. For the communication to happen, the parties must be connected to each other over a communication network. The approach to solve the problem of MPC will differ based on the properties of the underlying network. Below we discuss the various network models that are explored in the literature of MPC. A network consists of point-to-point channels through which the parties communicate. We assume that the channels between the pair of parties to be *secure* and *authenticated*. A secure channel ensures none other than those sitting in one of the end-points of the channel can see what is being transferred over the channel. An authenticated channel, on the other hand, ensures none can change the message transferred over the channel during its transit. For some scenarios, we will see that secure and authenticated channels are physical requirement from the network. While for many other scenarios, we will see how to emulate a secure and authenticated channel from insecure and unauthenticated channel relying on cryptographic primitives. Now we discuss the networks models that are studied

in the literature of MPC.

1.2.1 Complete vs. Incomplete

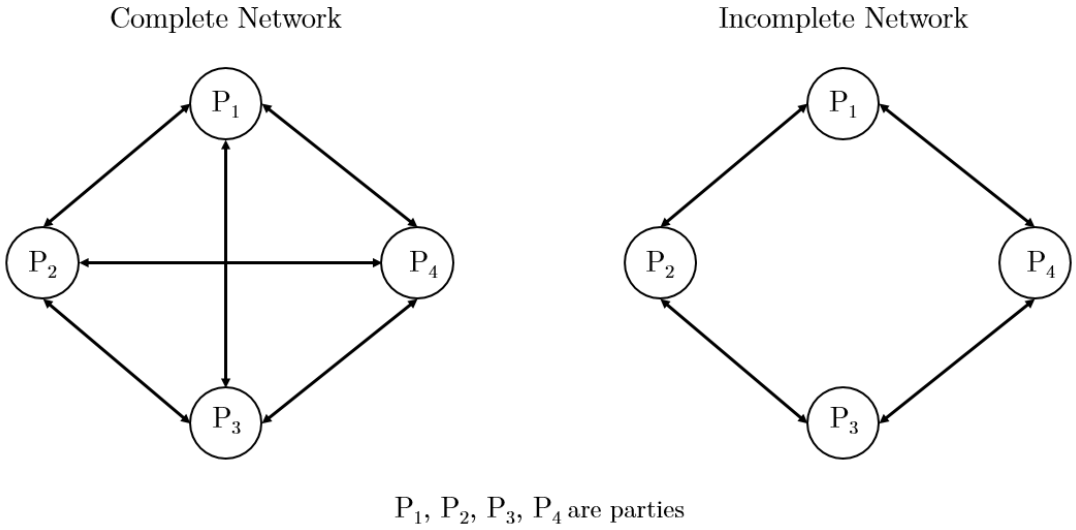

Figure 2: Complete vs. Incomplete Network

In a **complete** network, every pair of distinct parties is connected by a unique point-to-point channel. In contrary, point-to-point channel will be absent for some pair of parties in an **incomplete** network. The former model is considered to be standard in MPC literature. For MPC applications that require small number of participating parties, assuming a complete network is not a big constraint. Also MPC being a complex primitive in cryptography, the researchers like to start with a strong assumption on the underlying network (i.e. completeness of the network) and like to handle the problem of simulating a complete network from an incomplete network as a separate problem. However, for MPC applications that involves billions of participants (such as e-voting in a country), it is not realistic to assume the existence of a complete network. Therefore there is absolute need to study the incomplete network model. Unfortunately the latter model is very less explored and thus it leaves a huge scope of future work.

1.2.2 Synchronous vs. Asynchronous vs. Hybrid

In a **synchronous** network, all the parties have a global clock. Every channel in the network has a fixed delivery time. Consider a scenario with two parties. One of them is the sender who wants to send a message to the other party called a receiver. Due to the fixed delay of the channels, the receiver knows how long to wait for a message from the sender. If he is not receiving within the time period, he knows for sure that the sender is a cheater.

In an **asynchronous** network, there is no global clock. Each party has its own local clock and the local clocks are not synchronised. Hence the channel can have arbitrary, yet finite delay, meaning that if a message is sent over a channel, then it will reach *eventually*. We now review the same scenario that we considered for the synchronous network. In an asynchronous network, we will see that the receiver will face the following dilemma which he cannot resolve. When a receiver is not getting a message from the sender, he cannot conclude that the sender is bad. The delay can happen either due to a cheating sender or due to the traffic in the channel. Therefore, the receiver has no choice other than ignoring the message from the sender in order to terminate the protocol. Consider an n-party protocol in an asynchronous network. Let us assume that in the protocol, there is a step where a party is supposed to receive communication from every other party. Note that here unlike in synchronous network, the party cannot wait to receive values from all the parties. If it does so then the wait may turn out to be endless. The bad parties (say there are t in number) may never send their message. So the receiver party can wait to receive values from only $n - t$ parties and has to ignore values from t parties. But the $n - t$ parties that it considers may exclude t honest parties. Therefore, at any step of an n-party protocol, a party may have to proceed with the computation without the values from t honest parties. This makes constructing MPC protocol very challenging in asynchronous network. Most of the protocols in synchronous network simply breaks down in asynchronous network. To be specific, consider the secure addition protocol (given in Figure 3) for $n = 3$ parties discussed in the last lecture, with the number of corrupted parties, t is limited to 1. If we follow the

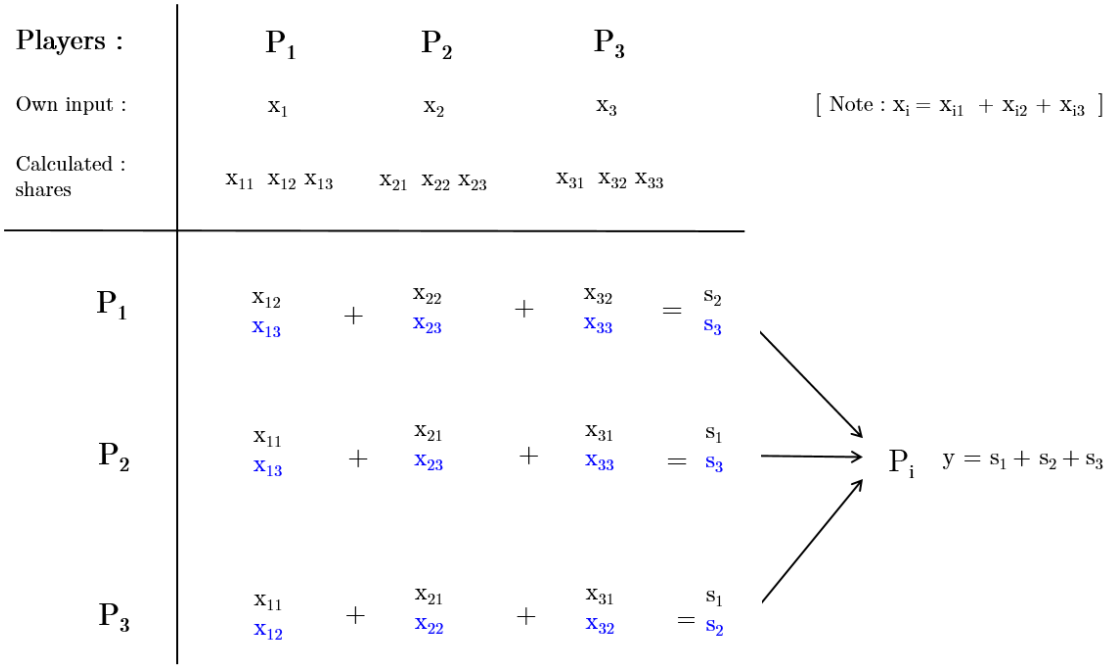

Figure 3: Secure addition for $n = 3$ parties

procedure described above, then each of the party will wait for messages (s_i in this context) from $n - t = 2$ parties and upon receiving, he has to proceed. But here, a party needs shares from all the three parties to compute the sum. Thus the protocol that will work in synchronous setting will simply fail in asynchronous network. In fact, it is proved that *no protocol with n parties, where t among them will be cheating, works in the asynchronous setting when $n \leq 3t$.*

There is a third category of network named **Hybrid** network. In this, the network is synchronous up to some point of time initially and afterwards it turns into an asynchronous network.

The synchronous network is well-behaved and thus well-understood and preferred in most of the instances, specifically in small local network. But real-life networks like Internet are better modelled by hybrid and asynchronous networks. Both the hybrid and asynchronous models are very less explored. Hybrid network though a weakening of asynchronous modelling, there are some impossibility results in asynchronous that are shown to be possible in Hybrid network. For instance, the above impossibility result of MPC with $n \leq 3t$ parties in asynchronous network can be overcome in hybrid network! So it makes sense to explore the hybrid network extensively.

1.3 Dimension 3: Modelling Distrust

Consider the secure addition protocol given in Figure 4. We know that this protocol is secure against a single curious party. To be precise, even if a corrupted party tries to get additional information out of the protocol, he cannot. But what if two parties are curious and they join hand? Then the protocol will fail. Specifically, consider the scenario where the parties P_1 and P_2 together are trying to cheat. Then P_1 can get his missing share x_{21} from P_2 and thus he can calculate x_2 (similarly for all the x_i's). The same can happen if any two among the four parties join hand. To model this idea that *bad people work together*, we assume that there is a single monolithic / centralized entity who we call as **adversary** ($\mathcal{A}$) and who controls a number of parties out of n parties. Note that it is always better to assume that the bad parties can collude with each other and provide security even in the face of a colluding adversary.

The scenario where the corrupted parties are working independently is termed as **decentralised** model and if the corrupted parties are working together, the model is termed as **centralised** model. In this course, we will consider only centralised modelling of the adversary. With the centralised adversary in mind, let us retouch the MPC definition.

1.3.1 Redefining MPC

In the centralised model, MPC can be redefined as follows: *There are n parties $P_1, P_2, ..., P_n$, out of which, say at most t are corrupted by an centralised adversary, $\mathcal{A}$. Each party P_i has its own private input x_i and there exist a common n-input function, f, which every party wants to compute. The goals of the MPC are as follows:*

- **Correctness :** Every honest party must output $y = f(x_1, x_2, ..., x_n)$.
- **Privacy :** Nothing beyond the output y is leaked to $\mathcal{A}$.

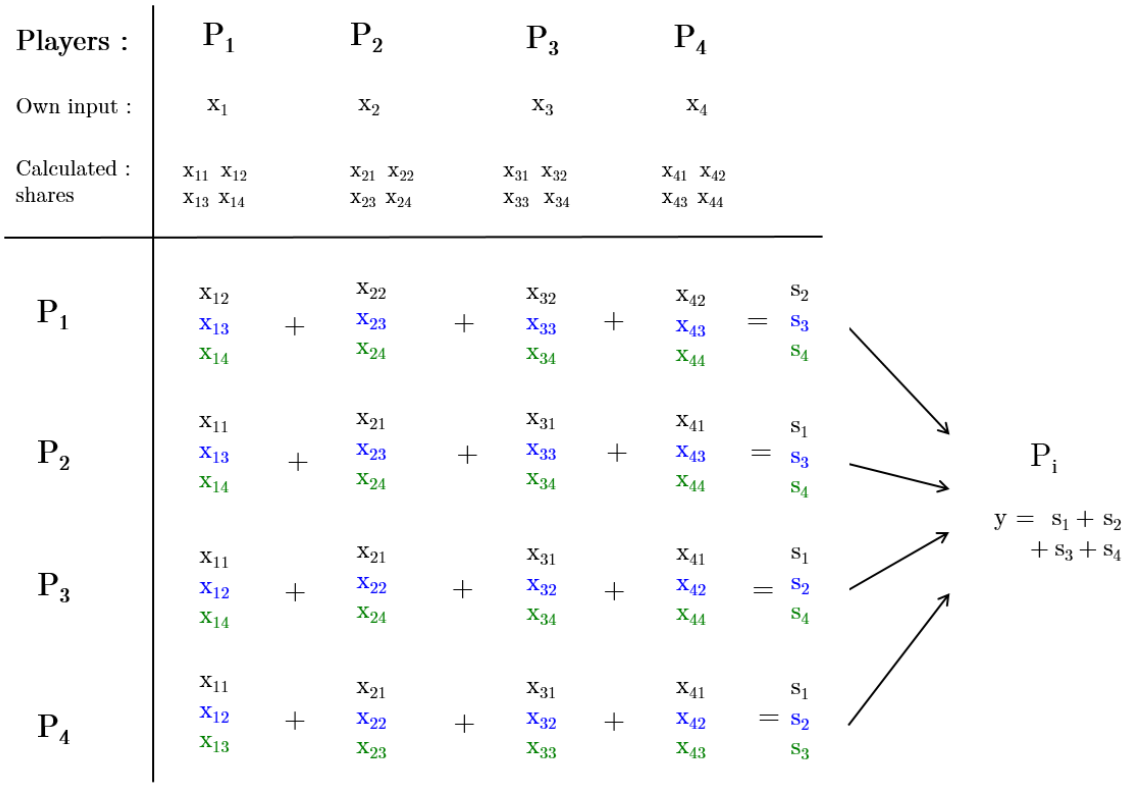

Figure 4: Secure addition for $n = 4, t = 2$ parties (Basic)

1.3.2 A quick Fix of Secure Addition Protocol of Figure 4

We have seen that the secure addition protocol presented in Figure 4 fails if there is a coalition of two parties. Can we quickly fix the protocol to make it work for adversary corrupting two parties? Yes, it is easy! We need to change the underlying secret sharing scheme. The changed protocol is given in Figure 5.

1.4 Dimension 4: Characteristics of Adversary

There are several types of adversaries depending on its capabilities or intentions they are presumed to have. We consider four orthogonal dimensions and in each dimension, the adversary can be categories into several types.

1.4.1 Threshold vs. Non-threshold

In the **threshold** model, the adversary $\mathcal{A}$ can corrupt at most t out of the n parties. Here n denotes the total number of participating parties and t represents the *threshold*. Note that $t < n$. In **non-threshold** model, the adversary's behaviour is captured by a set of subset of parties. The adversary $\mathcal{A}$ can corrupt one of the subsets.

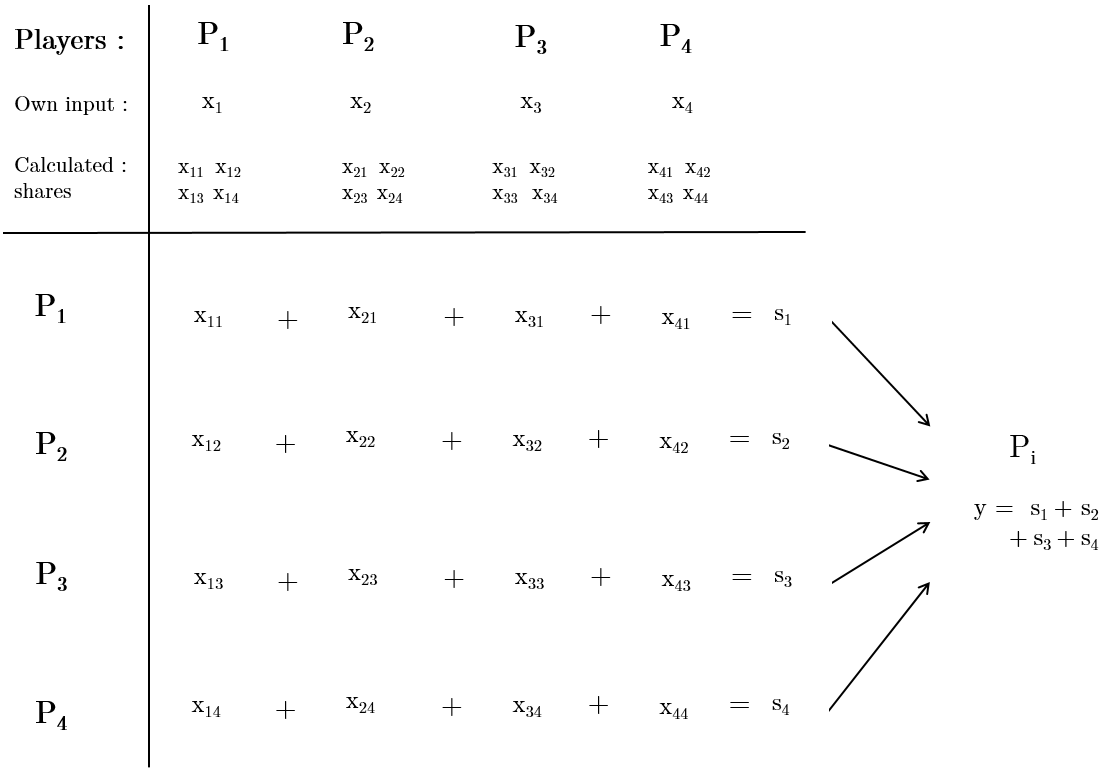

Figure 5: Secure addition for $n = 4, t = 2$ parties

Let $P = \{P_1, P_2, P_3\}$ denote the set of players. Then a corrupting set $C = \{\{P_1\}, \{P_2, P_3\}\}$ means the adversary, $\mathcal{A}$ can corrupt either the player P_1 or the players P_2 and P_3. The threshold model can be viewed as a special case of the non-threshold model. Most of the works in MPC assume threshold adversary because of its simplicity. The non-threshold adversary models real-life scenario much better way than threshold adversary, but it is hard and non-intuitive to design protocol tolerating such an adversary. We note that *non-threshold secret sharing*, where secret is shared to parties in such a way that only legitimate subset of parties are able to reconstruct the secret is a very rich topic of research.

1.4.2 Polynomially Bounded vs. Unbounded

Intuitively, a **polynomially/ computationally bounded** adversary is an algorithm that runs in polynomial amount of time. Here the adversary's computing power is limited to $\mathrm{O}(n^c)$ where n is the input size and c is some constant. For a cryptographic scheme to be secure, the success probability of any polynomial time adversary must be *negligible*. In order to prove security in this model, we rely on hard number-theoretic problems. The proofs are usually *reduction based* and works as follows: Assume our protocol is not secure and we have an adversary, say $\mathcal{A}$, that breaks the protocol with non-negligible probability. We will use $\mathcal{A}$ to come up with another adversary, $\mathcal{A}'$, which can break some known hard problems. This will lead to a contradiction of the fact that the hard problems are not hard!

This in turn proves that the protocol under consideration is secure. The security that is achieved in this setting is often terms as *computation or cryptographic security.*

An **unbounded powerful adversary** has got no limitation to its computing power. Here the protocols security does not rely on any hard problem assumptions and the security is very strong since it holds even at the face of all-powerful adversary. Even if $\mathcal{A}$ has got quantum computers, it cannot break the security. The security that is achieved in this setting is often terms as *information-theoretic (i.t.) security.* Now a question that comes to your mind is how such a security can be achieved? Consider the simple experiment of tossing an unbiased coin where the adversary, $\mathcal{A}$, is trying to predict the next output. Without any additional knowledge or computation, the probability of $\mathcal{A}$'s prediction being correct is $1/2$. Even if $\mathcal{A}$ is possessing the output of some large number of previous experiments, it cannot predict the next output with a probability $> 1/2$. This is just an example to get familiarized with this information theoretic setting. Nonetheless, i.t. security is impossible to achieve in the *dishonest majority ($n < 2t$)* setting (except for some trivial functions). Secure bit multiplication is one example that cannot be computed with i.t. security in dishonest majority setting.

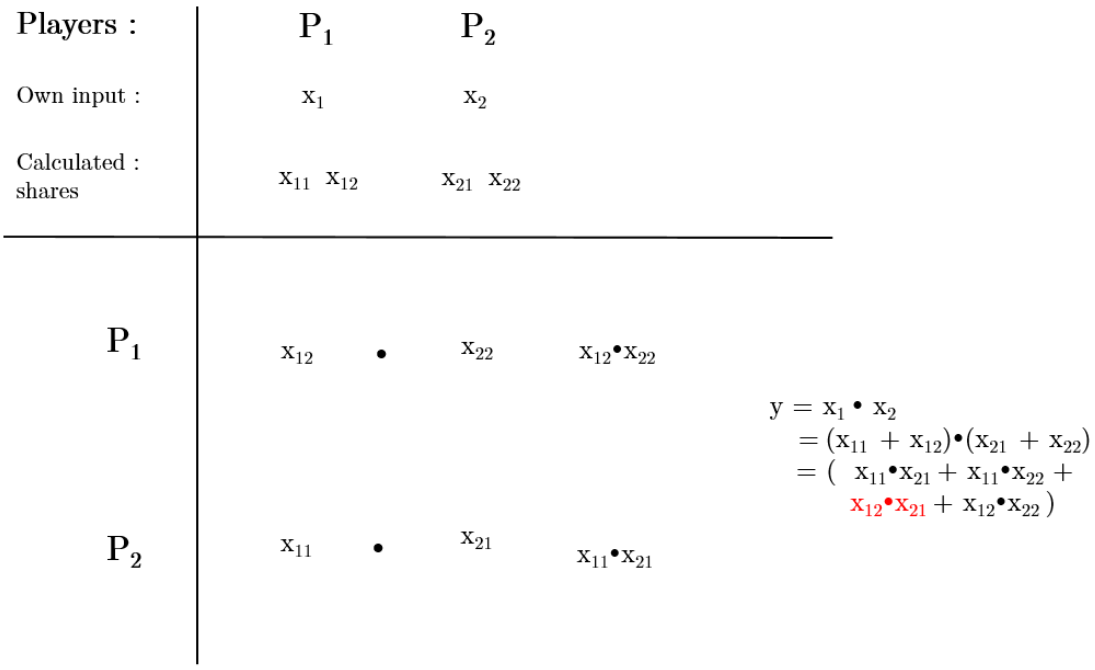

Figure 6: Secure Multiplication for $n = 2, t = 1$ parties

Recall the Secure Multiplication protocol 6 for finding $y = x_1 \cdot x_2$ with $n = 2$ and $t = 1$. The problem lies in calculating $x_{12} \cdot x_{21}$ securely. We used *Oblivious Transfer (OT)* primitive to solve the problem. But OT cannot be realized information theoretically too!

But we can solve the problem with i.t. security with 3 parties where one of the is corrupted (that is, when we have honest majority). As shown in Figure 7, each of the party, P_i can calculate s_i and as you can see, the sum of s_i's yields the output y. But is this

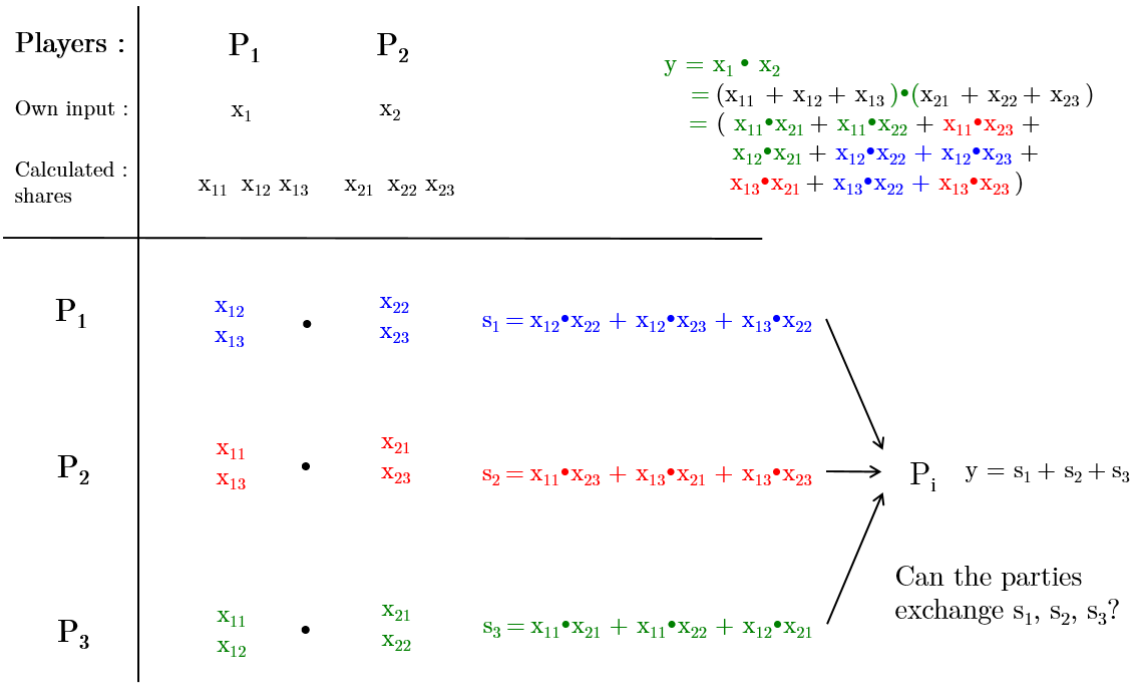

Figure 7: Secure Multiplication for $n = 3, t = 1$ parties

protocol secure? Without loss of generality, let P_1 be corrupted. Then he will obtain x_2 as follows: P_1 possesses the shares x_{22} and x_{23}, and all he needs is x_{21}. Now, s_3 can be written as $s_3 = (x_{11} + x_{12}) \cdot x_{21} + x_{11} \cdot x_{22}$, where P_1 knows s_3, x_{11}, x_{12} and x_{22}. Thus exchanging the s_i values will not work. But we have a simple solution. We can make use of the Secure Addition Protocol [3] to obtain y, where the values s_1, s_2, s_3 act as secret inputs of parties P_1, P_2 and P_3 respectively.

1.4.3 Semi-honest vs Malicious vs Covert

A **passive/semi-honest** adversary simply acts as an observer. It eavesdrops the corrupted parties and tries to gain more information than allowed from the protocol transcript. But it follows the prescribed protocol. In contrary, an **active/malicious** adversary takes full control over the corrupted parties. As such it can deviate at will from the prescribed protocol. There is a third category, called **covert** adversary where the adversary may behave maliciously only when its probability of getting caught is low. Usually the protocols secure against the semi-honest adversaries will not be secure in the malicious model. Defeating malicious adversaries demands a whole lot of new primitives like commitment schemes, zero-knowledge proofs, Byzantine agreement/broadcast etc. The semi-honest model will be considered in the first half of the course and the the second half will deal with malicious adversary. Let us now review some of the secure protocols that we have seen. We will see that they work *only* when the adversary is semi-honest! Consider the Secure Addition protocol shown in Figure 8. Without loss of generality, let P_1 be the corrupted party. One

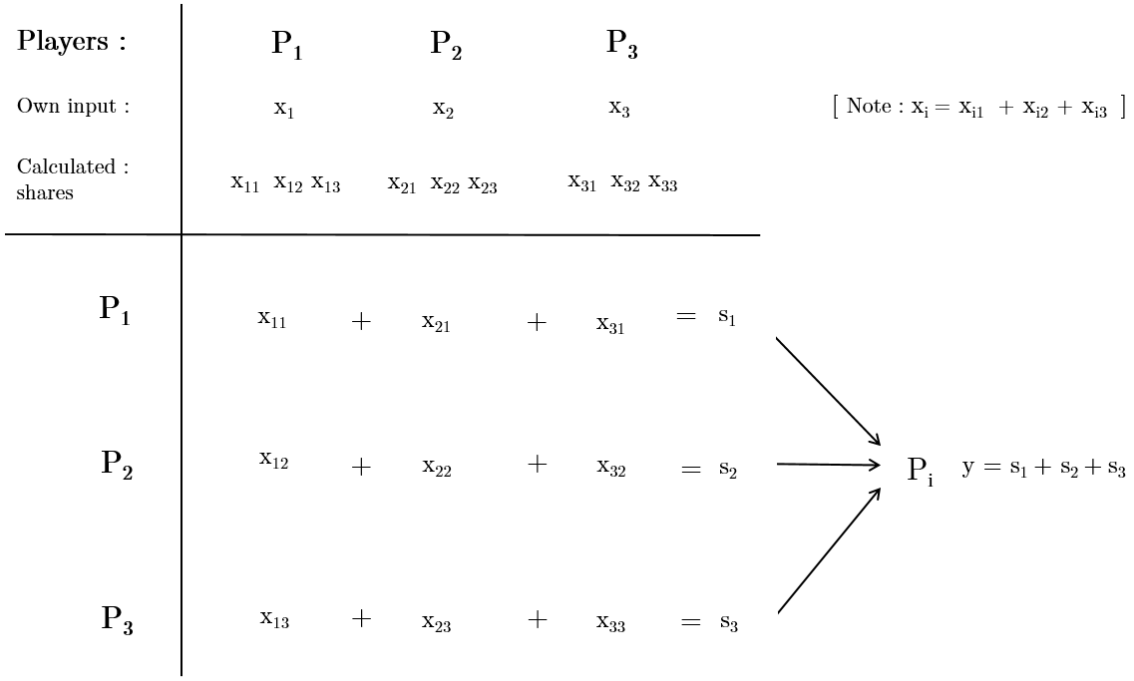

Figure 8: Secure Addition for $n = 3, t = 1$ parties - Malicious setting I

simple thing that P_1 can do to make the protocol fail is not to send his share s_1. This will make both the honest parties (P_2 and P_3) agree on some $\perp$ as they will not be able to complete the protocol. But P_1 can do even worse. Consider the protocol shown in Figure 9. We will show a simple strategy of $\mathcal{A}$ by which he can make P_2 and P_3 output different sums or outputs. P_1 distributes his share s_1 as follows: P_1 will send s_1 to P_2 and some other $s_1{}'$ to P_3. This will make P_2 output y and P_3 output some other y'.

What can we do to solve this problem? One simple solution that comes to mind is redistribution of the shares. Every player will distribute the computed output, y, with each other. Now a party will check whether all the y values he received is matching with his output or not. If all of them matches, he will accept his output, else accept on $\perp$. This seems to solve the problem we discussed earlier.

Consider the following strategy of $\mathcal{A}$. P_1 will act in accordance with the protocol until the final redistribution. While distributing the output values, P_1 will share the correct output with P_2 and some other value with P_3. Thus P_2 will accept his output since all the values he received is matching with his output, but P_3 will output $\perp$ since its output does not match with other's output. Even if we include similar redistribution phase a number of times, the adversary will follow the strategy mentioned above. Thus this change will fail to save the protocol. What we need to make the parties to agree on the output is yet another interesting primitive called broadcast or Byzantine agreement (BA) that we discuss below.

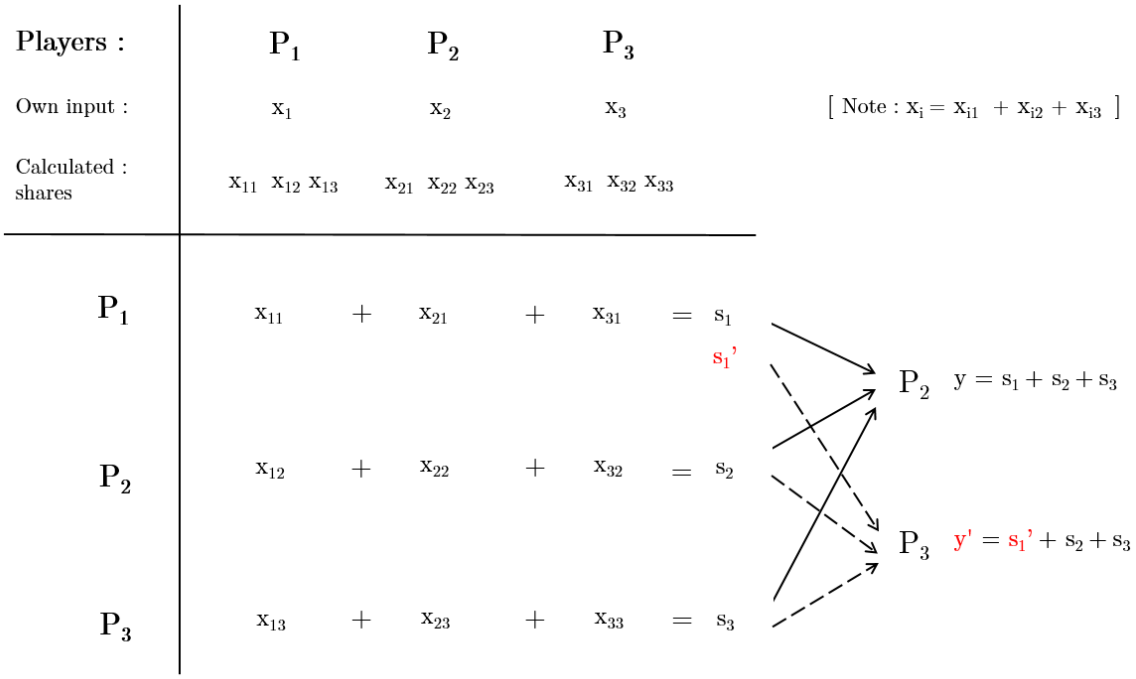

Figure 9: Secure Multiplication for $n = 3, t = 1$ parties - Malicious setting II

Broadcast. [ii] The **broadcast** primitive is defined as follows:There are n parties $P_1, P_2, ..., P_n$, out of which t are corrupted by an adversary, $\mathcal{A}$. Parties are connected by pair-wise secure and authenticated channels. There is a designated party named Sender, who wants to send his message m to every party.The goals of the protocol are as follows:

- All the honest parties must receive the some message, say m', at the end of the protocol.
- If the Sender is honest, then $\mathsf{m}' = \mathsf{m}$.

We can now solve the issue of disagreement of the outputs in the protocol in Figure 9 in the malicious setting. The protocol is modified using *Broadcast* primitive and is shown in Figure 10. Here every party broadcast the calculated s_i values, so that a cheating party cannot send different values to honest parties. In this protocol, even though there is *agreement* among the honest parties, no *robustness* or *fairness* is guaranteed. Here P_1 may not broadcast his share, s_1, while he receives s_2 and s_3 from the other parties. Thus adversary learns output y, while the honest parties do not. This is an unfair situation. Also the protocol cannot proceed without the share s_1. Thus the protocol lacks robustness, which can be defined as the ability of an algorithm to cope with errors during execution.

A related problem of broadcast known as Byzantine Agreement is defined below.

[ii] Topics from this section onward was covered in the combined Lecture 3 & 4 on August 21, 2015.

Byzantine Agreement (BA). The **Byzantine agreement** problem can be defined as follows: *There are n parties $P_1, P_2, ..., P_n$, out of which t are corrupted by an adversary, $\mathcal{A}$. Parties are connected by pair-wise secure and authenticated channels. Each party P_i has a private bit* $\mathsf{b_i} \in \{0, 1\}$. The goal of the protocol is to make the honest parties agree on a common bit b.

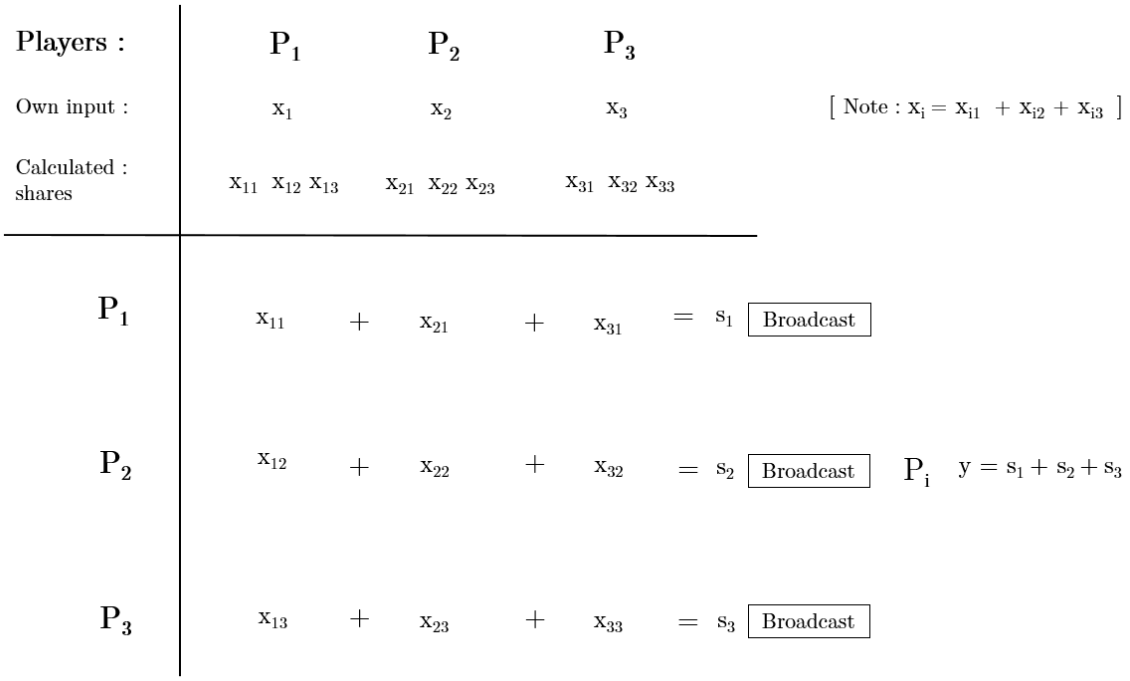

Figure 10: Secure Addition for $n = 3, t = 1$ parties using Broadcast (Malicious setting)

Next we define two more primitives that find application in malicious setting.

Commitment Schemes. Consider the problem of *2-party distributed coin tossing* in Figure 11. Two players A and B want to toss a coin together. There will be a value associated with each outcome. Without loss of generality, say 0 for *head* and 1 for *tail*. Let the outcome of A and B be m_A and m_B respectively. Now the goal of the problem is to agree on $m_A + m_B$. One naive method is to toss the coin independently and share the results. Here what if B is bad? He will then choose his m_B based on the value of m_A received. This defeats the purpose.

Commitment schemes (refer to Figure 12) can solve the above problem. Two parties are involved in a commitment scheme - *Committer* and *Verifier*. In Figure 12, Alice (Committer) first commits her message m, obtains the commitment C and sends to Bob (Verifier). At a later point, Alice sends her message m, along with commitment opening information, so that Bob can verify. This can be compared with a lock and key mechanism. First the message is locked in a box and is sent. Here the box acts as the commitment. Later the key, which acts as the opening information, along with the message is sent to the verifier.

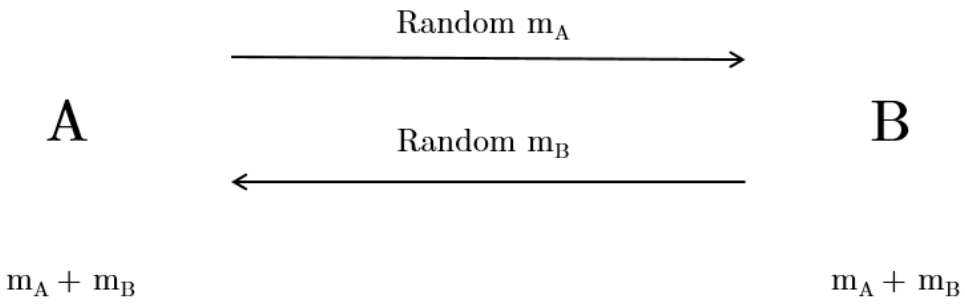

Figure 11: 2-party distributed coin tossing I (Malicious setting)

Two desirable properties of commitment schemes are:

- **Hiding :** Committer (Alice) cannot change the message associated with the commitment, C.
- **Binding :** Verifier (Bob) cannot guess the message associated with the commitment, C.

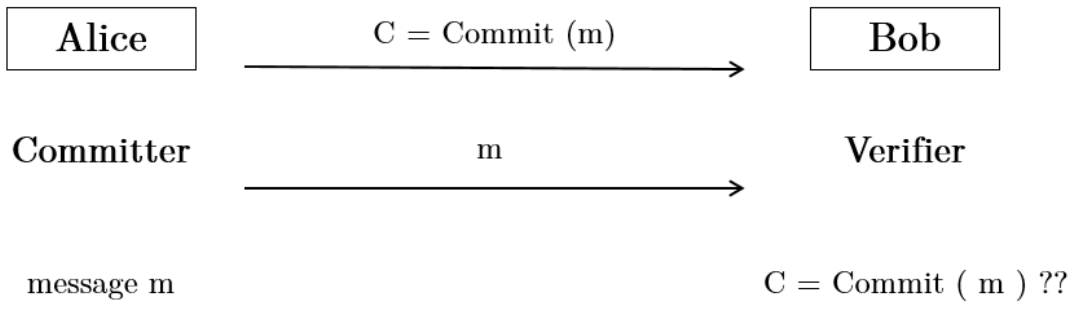

Figure 12: Commitment Scheme

Figure 13 shows the 2-party distributed coin tossing protocol using commitment scheme. Here B cannot bias the output due to the *hiding* property of the commitment.

Zero-knowledge Proofs. The purpose of a traditional proof is to convince somebody, but typically the details of a proof give the verifier more info than the assertion that the

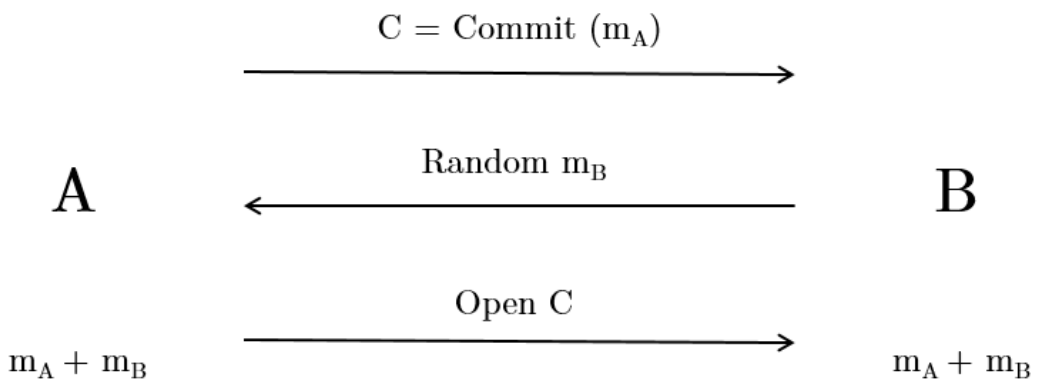

Figure 13: 2-party distributed coin tossing using commitment (Malicious setting)

given statement is correct. A **zero-knowledge proof** is a method by which one party (the *prover*) can prove to another party (the *verifier*) that a given statement is true, without conveying any information apart from the fact that the statement is indeed true. The zero-knowledge proofs are of extraordinary importance in cryptography.

1.4.4 Static vs Adaptive

A **static** adversary needs to decide the set of players to corrupt prior to the execution of the protocol, while an **adaptive** adversary can corrupt players during the execution of the protocol arbitrarily. Adaptive adversary is more flexible, realistic and stronger than the static corruption, since it gives the adversary more power to selectively attack the parties after seeing the communication. In addition to above two, there are other classifications like *Semi-adaptive, One-sided Adaptive and Partial-erasure Adaptive* which are out of scope of this course and fall in between the static and adaptive modelling.

1.5 Expanding the scope of MPC: Summary

So far, we have seen four major dimensions of MPC - *Models of Computation, Network, Distrust Model, Adversary.* Each of these dimension contains further classification as shown in Table 1. As you can see, each of these dimension is independent of the other. Before working on any MPC problem, you have to first clearly define the *setting.* If the adversary's behaviour is unknown, then we can only reason about security very informally. Also it is not clear what is and what is not protected against.

For example, *a complete synchronous network controlled by a centralized adversary who is polynomially-bounded, semi-honest and static with the threshold model* is one such a setting. From Table 1, if we consider one model from each of the dimensions, we can have

$2 \cdot 5 \cdot 2 \cdot 9 = 180$ different settings. We can have many more such setting and the saga of MPC continues...

Dimension 1 (Models of Computation)	**Dimension 2** (Networks)	**Dimension 3** (Distrust)	**Dimension 4** (Adversary)
Boolean vs. Arithmetic	Complete vs. Incomplete Synchronous vs. Asynchronous vs. Hybrid	Centralized vs. Decentralized	Threshold vs. Non-threshold Polynomially Bounded vs. Unbounded Powerful Semi-honest vs. Malicious vs. Covert Static vs. Adaptive

Table 1: Expanding the scope of MPC: Summary (Underlined models will not be covered in this course)

Next we will see what are the important attributes of an MPC protocols and what are the type of questions asked in the context of MPC.

2 Attributes of MPC protocols

An MPC protocol is always associated with a set of attributes which can be used to compare and contrast among various MPC protocols solving the same problem. They are:

Parameter 1 (Resilience). Resilience os a protocol is nothing but the the number of corrupted parties (t) among the n parties which the protocol can tolerate. Common settings for MPC include $n > 4t$, $n > 3t$, $n > 2t$ and $n > t$.

Parameter 2 (Quality). Qualitatively a protocol can be of following types:

1. **Perfect (error-free)/ Statistical:** A perfect protocol tolerates no error, while a statistical one can tolerate small amount of error. Statistical security is usually easer to achieve than perfect security.

2. **Robust / Non-robust:** A protocol is robust if the adversary cannot influence output of the honest parties in the protocol.

3. **Fair / Unfair:** In a fair setting, either all the parties (honest and corrupted) get the output or no one gets anything, while in an unfair setting, the corrupted parties learn the output, while the honest parties do not.

Parameter 3 (Complexity). An MPC protocol will have the following complexity measures:

1. **Communication Complexity:** Total number of bits communicated by the honest parties.
2. **Round Complexity:** Total number of rounds of interaction in the protocol.
3. **Computation Complexity:** Computation time required for running protocol.

The complexities of a protocol becomes crucial mainly when the protocol comes into practice. The ultimate goal is to construct protocols with optimal complexities.

3 Questions in MPC

Based on different settings, one can ask different questions regarding the feasibility, efficiency, optimality and so on.

Question 1 (Possibility/Impossibility): *Given the network type and adversary type, under what conditions MPC is possible?* Some of the standard results are:

a Information theoretic MPC is possible iff $n > 2t$

b In synchronous networks, perfect (i.t) MPC is possible iff $n > 2t$

c In asynchronous networks, statistical (i.t) MPC is possible iff $n > 3t$

d In asynchronous networks, perfect (i.t) MPC is possible iff $n > 4t$

e In synchronous networks, computational robust fair MPC is possible iff $n > 2t$

Question 2 (Efficiency): *Given the network type and adversary type, how efficient (communication/round/computation) MPC can be designed?*

Question 3 (Optimality): *Given the network type and adversary type, what is the optimal complexity we can achieve? Design such optimal protocols.*

4 Security of MPC

Recall the definition of MPC in Section 1.3.1 we discussed earlier. Does that definition captures all the needs? It doesn't. Consider the problem of *Secure Auction* (with secret bids). We try to analyze the requirements for the protocol.

- An adversary may wish to learn the bids of all parties - to prevent this, we require **PRIVACY**.
- An adversary may wish to win with a lower bid than the highest - to prevent this, we require **CORRECTNESS**.
- But, the adversary may also wish to ensure that it always gives the highest bid - to prevent this, we require **INDEPENDENCE OF INPUTS**.
- An adversary may try to abort the execution if its bid is not the highest - we require **FAIRNESS**.

Some of the desirable properties of every MPC protocol are :

1. **Privacy:** The output is revealed and nothing more.
2. **Correctness:** The function is computed correctly.
3. **Independence of inputs:** Parties cannot choose inputs based on inputs of others.
4. **Fairness:** If a corrupted party (or adversary) receives output, honest parties also receive output.
5. **Guaranteed output delivery:** No matter how the corrupted parties behave, honest parties must get output.

Do you think the above list is complete? There are generally two methods by which we can analyze the security concerns. One simple option is to *analyze security concerns for each specific problem*, similar to what we did for Secure Auction above. But there are problems associated with this approach. First of all, how can we ensure that all the concerns are covered? Our considered list may not include all the concerns! Secondly the definitions will become application dependent and thus we need to find a definition each time we come across a new problem. This does not look really a nice approach! In what follows, we will formulate a definition paradigm that we hold across all applications of MPC meaning that across all the functions PPT functions f and clearly specifies that it captures- real world/Ideal world based Security paradigm.

4.1 Real World/ Ideal World Based Security

How do you judge a person (person's particular quality)/ a product? We set a *standard/ideal* and find out how close are we to the ideal. We will do exactly the same for MPC. First we set an ideal/standard/benchmark for MPC and then define security based on the closeness to the ideal solution.

Can you think of a solution that can act as an ideal solution or benchmark solution for MPC. In fact we have seen one potential solution in the first lecture itself. Remember solving MPC with the help of a *trusted third party (TTP)*! The parties send inputs to the TTP, who computes the function and sends the outputs to the parties over secure channels. The

TTP is connected to the parties with secure (none can see what TTP sends to the honest parties) and authenticated channels and there are no communication channels among the parties. This is 'the' ideal solution for MPC. We will call this as *Ideal world protocol*. Note that if the adversary $\mathcal{A}$ corrupts a party in the ideal world protocol, it learns only the input and output of the corrupted party and nothing beyond. Since it learns nothing beyond, independent of inputs is guaranteed apart from privacy. We can see that how precise the specification and description of the ideal world. It is well-understood. We know what all properties it provides in an obvious way and can change the specification according to our need.

In the *real world*, parties run a real protocol with no trusted help and tries to emulate the ideal world. Consider a setting where there are n parties connected over a complete synchronous network and there is a centralized adversary who is polynomially-bounded, semi-honest and static and the corruption threshold is t. The parties communicate among themselves and finally obtains their corresponding output.

In Figure 14, we specify the real world and the ideal world solution for MPC assuming 4 parties out of which at most 1 party can be corrupted. Each party, P_i has his private input, x_i and together they want to calculate a function f and to obtain the respective input y_i.

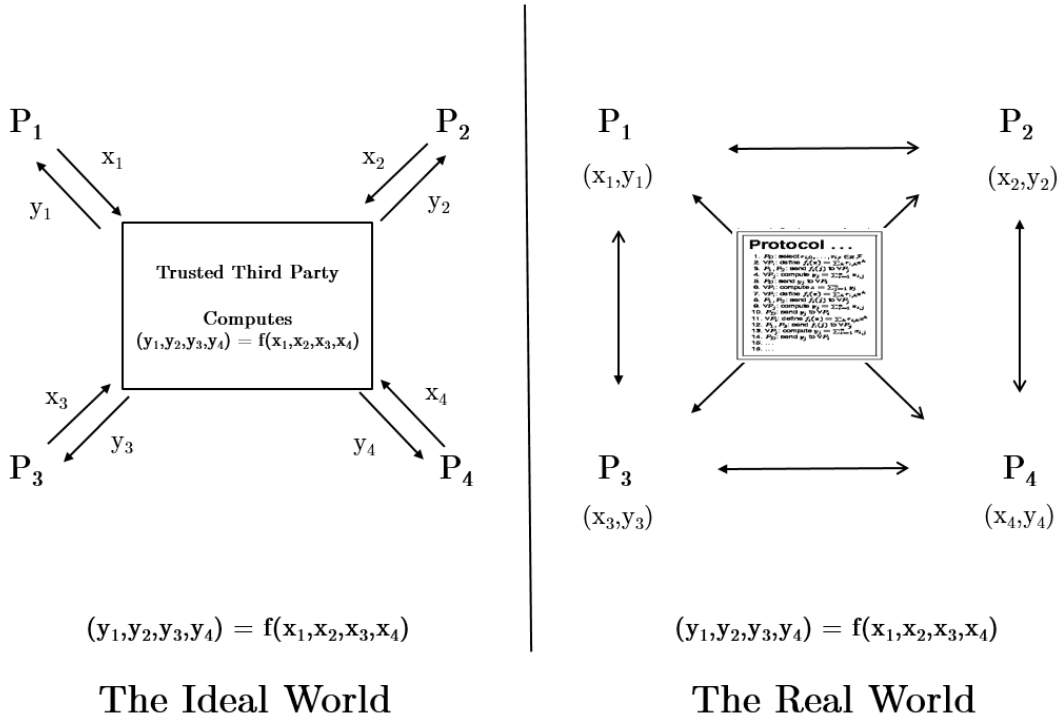

Figure 14: Real world vs Ideal World

Now how do you compare real world with the ideal world solution? How can we say that our real world emulates closely the ideal world solution? Lets us fix the inputs of the parties as $(x_1, \ldots, x_n)$ used in both the worlds to compute $f(x_1, \ldots, x_n)$. We can say that

if the real world view of the adversary contains no more information than its ideal world view, then the real world is as good as the ideal world. Sounds good!

Let $\mathsf{View}_i^{\mathsf{Ideal}}(x_1, \ldots, x_n))$ and $\mathsf{View}_i^{\mathsf{Real}}(x_1, \ldots, x_n))$ denote the views of a party, P_i, in the ideal and real world protocols on the inputs $x_1, x_2, \cdots, x_n$. Let C denote the set of corrupted parties. Then view of the adversary, $\mathcal{A}$, is defined as the combined view of all the parties in C. That is,

$$\begin{aligned} \mathsf{View}_{\mathcal{A}}^{\mathsf{Ideal}}(x_1, \ldots, x_n) &= \{\mathsf{View}_i^{\mathsf{Ideal}}(x_1, \ldots, x_n)\}_{P_i \in C} \text{ and} \\ \mathsf{View}_{\mathcal{A}}^{\mathsf{Real}}(x_1, \ldots, x_n) &= \{\mathsf{View}_i^{\mathsf{Real}}(x_1, \ldots, x_n)\}_{P_i \in C} \end{aligned}$$

Note that the view of a party in the ideal world constitutes of only the input and the output of that party. Whereas the view in the real world includes the the entire protocol transcript, the random bits used apart from the input and outputs. The values in the ideal world view are termed *allowed values* (for MPC these are the values we can leak to the adversary, that is why the name allowed values), while those in real world are termed *leaked values*. We say that *our protocol is secure if the leaked values contains no more info than allowed values*. But what does it mean by a value, say Y contains no more information than a value, say X? If there exists a polynomial time algorithm that takes X and can generate Y efficiently, then we will say that Y contains no more information than X. We call such

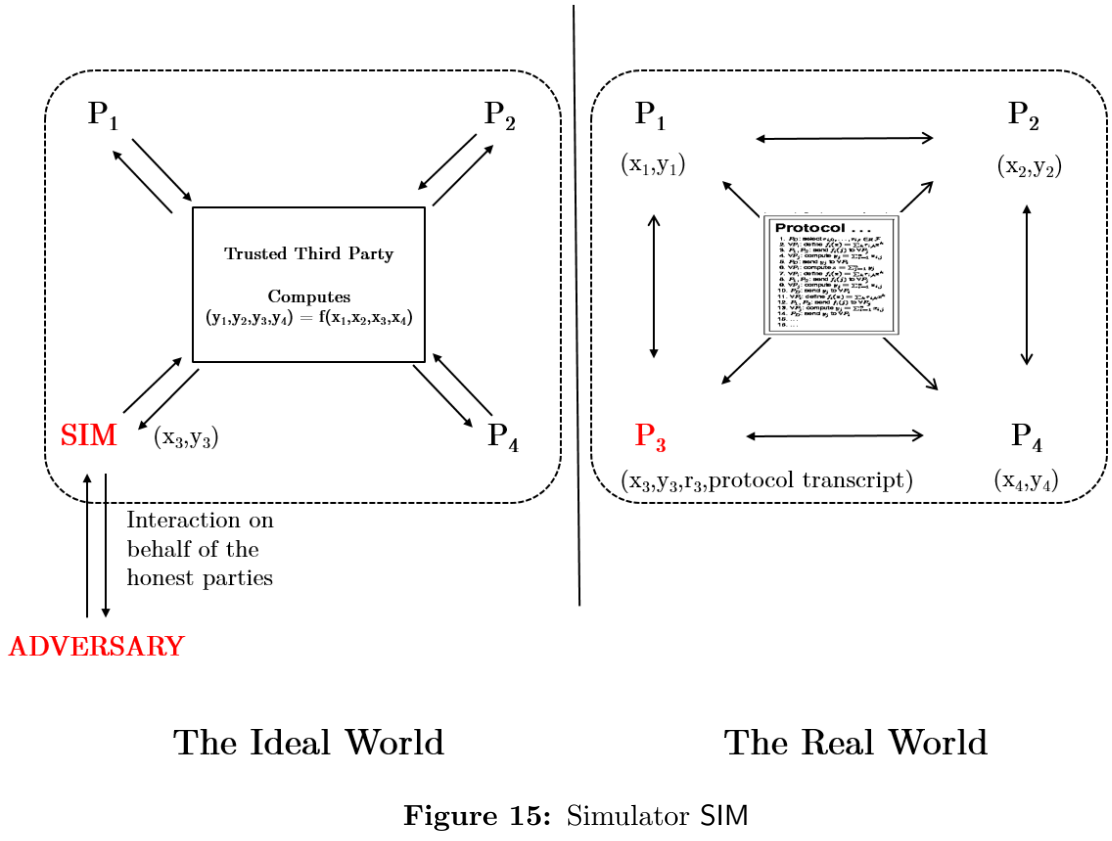

Figure 15: Simulator SIM

a poly-time algorithm as simulator SIM (refer to Figure 15). For our purpose it is enough, if SIM can generate a view that resembles 'close enough' $\mathsf{View}_{\mathcal{A}}^{\mathsf{Real}}(x_1, \ldots, x_n)$ starting with

the allowed values $\mathsf{View}_{\mathcal{A}}^{\mathsf{Ideal}}(x_1, \ldots, x_n)$. By 'close enough' means the the output of SIM and $\mathsf{View}_{\mathcal{A}}^{\mathsf{Real}}(x_1, \ldots, x_n)$ must be indistinguishable from the point of view of the adversary $\mathcal{A}$.

Such a simulator can simulate the real view of the $\mathcal{A}$ in ideal world by acting on behalf of the honest parties but without any access to the inputs of the honest parties. All it takes as inputs are the inputs and outputs of $\mathcal{A}$ which are the allowed values. The simulator essentially acts as a translator from the ideal world protocol to between the real world protocol. Now look carefully that in the real world, the view of the adversary is clearly function of the inputs of the honest parties. Whereas in the ideal world, the simulator simulates the view of the adversary but without any knowledge of the inputs of the honest parties. If the adversary sees no difference or it cannot tell apart in which which he is in, then we can say that our protocol employed in the real enough is secure and does not leak more than what the adversary sees in the ideal world (i.e. the allowed values). So we formulate our definition of secure as follows:

Definition 1 A protocol Π is secure according to Real World / Ideal World based security if for every probabilistic polynomial-time adversary $\mathcal{A}$, there exists a probabilistic polynomial-time simulator SIM such that for every possible inputs of the parties $(x_1, \ldots, x_n)$, the view of the adversary in the real world is computationally indistinguishable from the view of the adversary in the ideal world simulated by the simulator.

$$\mathsf{View}_{\mathcal{A}}^{\mathsf{Ideal}}(x_1, \ldots, x_n) \;\approx\; \mathsf{View}_{\mathcal{A}}^{\mathsf{Real}}(x_1, \ldots, x_n)$$

We abuse notation $\mathsf{View}_{\mathcal{A}}^{\mathsf{Ideal}}(x_1, \ldots, x_n)$ to denote the simulated view of the adversary in the ideal world with the help of the simulator. $\diamondsuit$

Note that the views in the above definition are random variables (distributions). The real view is a distribution over the random choices made by the parties during the run of the protocol. The simulated view is a distribution over the random choices made by the simulator. We would like to comment that when the adversary is semi-honest, then the entire simulation can be run by the simulator by self in its head. On knowing the inputs and outputs of the corrupted parties, it can initiate the corrupted parties with the inputs and it can further fix the randomness to be used by the corrupted parties. Then it can simulate the view the corrupted parties by acting on behalf of the honest parties as in the real protocol.

In order to capture randomized functions, we consider a joint distribution of the output of honest parties and the view of the adversary. Let H denote the set of honest parties and $\mathsf{Output}_i^{\mathsf{Ideal}}$ and $\mathsf{Output}_i^{\mathsf{Real}}$ denote the output of party, P_i, when he is honest and the inputs of the parties are $x_1, x_2, \cdots, x_n$ in the Ideal and Real world respectively. For this case the above definition has to be modified with the following equation:

$$[\ \mathsf{View}_{\mathcal{A}}^{\mathsf{Ideal}},\ \{\mathsf{Output}_i^{\mathsf{Ideal}}\}_{P_i \in H}] \;\approx\; [\ \mathsf{View}_{\mathcal{A}}^{\mathsf{Real}},\ \{\mathsf{Output}_i^{\mathsf{Real}}\}_{P_i \in H}]$$

Since any randomized function can be written as a deterministic function by fixing the randomness to be used, it is enough to have the first definition for deterministic functions. E.g. $g(x_1, x_2; r_1 + r_2) = g((x_1, r_1), (x_2, r_2))$.

References

[1] Ronald Cramer, Ivan Bjerre Damgrd, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing - An Information Theoretic Approach*. Cambridge University Press, 2015.

[2] Arpita Patra. *http://drona.csa.iisc.ernet.in/ arpita/SecureComputation15.html* . Course Materials.

[3] Yehuda Lindell, IBM T.J.Watson. *A tutorial on Secure Multiparty Computation*.