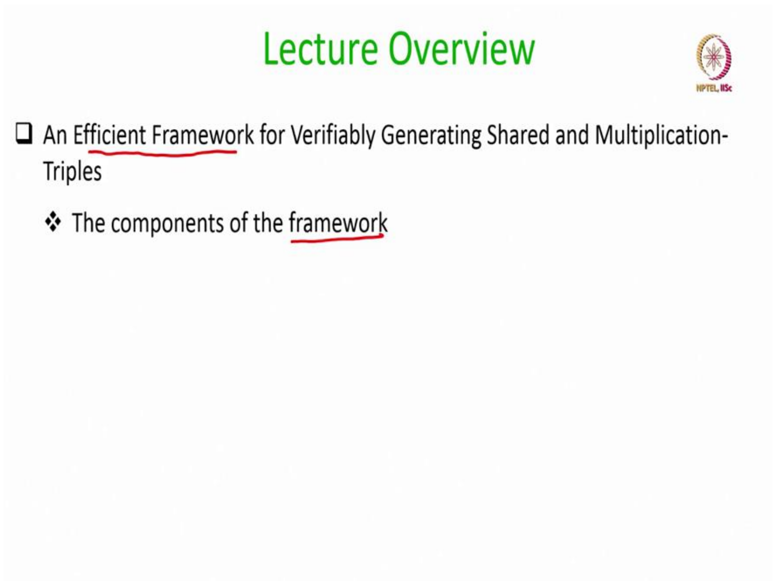
Secure Computation: Part II

Prof. Ashish Choudhury
Department of Computer Science and Engineering
Indian Institute of Science, Bengaluru

Lecture - 48
Generating Random Multiplication-Triples: I

Hello everyone welcome to this lecture.

(Refer Slide Time: 00:26)



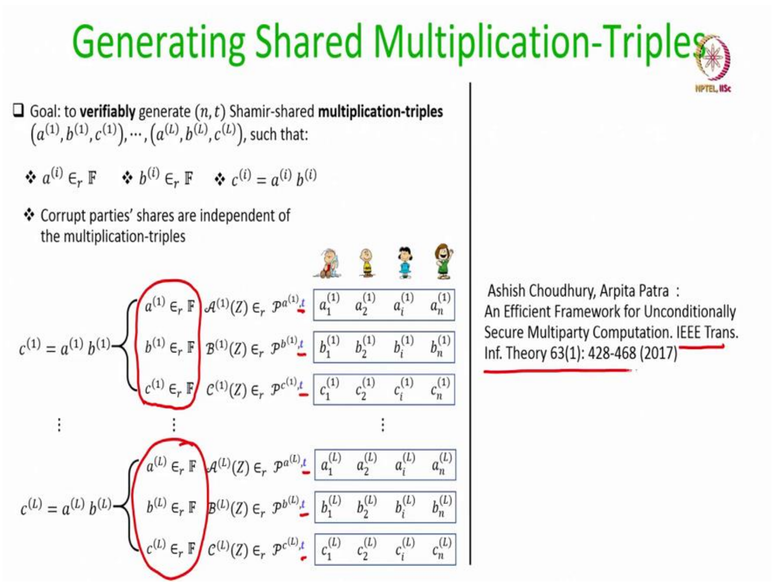
Lecture Overview

- An Efficient Framework for Verifiably Generating Shared and Multiplication-Triples
  - The components of the framework

So, in the last lecture we had seen the Beaver's method for solving the degree reduction problem, assuming that the parties have generated randomly secret shared multiplication triplets in the pre processing phase. Now, for the next few lectures our focus will be on the pre processing phase protocol. Namely how exactly we enable the parties to generate those secret shared random multiplication triplets.

So, for that we will discuss a very efficient framework for verifiably generating shared random multiplication triplets which is slightly involved. And we will slowly discuss each and every component of the framework. So, in this lecture we will discuss the main components of the framework and then in the subsequent lectures we will see how to instantiate those components.

(Refer Slide Time: 01:20)



Generating Shared Multiplication-Triples

- Goal: to **verifiably** generate (n, t) Shamir-shared **multiplication-triples** $(a^{(1)}, b^{(1)}, c^{(1)}), \cdots, (a^{(L)}, b^{(L)}, c^{(L)})$, such that:
  - $a^{(i)} \in_r \mathbb{F}$
  - $b^{(i)} \in_r \mathbb{F}$
  - $c^{(i)} = a^{(i)} b^{(i)}$
  - Corrupt parties' shares are independent of the multiplication-triples

$$c^{(1)} = a^{(1)} b^{(1)} \begin{cases} a^{(1)} \in_r \mathbb{F} & \mathcal{A}^{(1)}(Z) \in_r \mathcal{P}^{a^{(1)}, t} & \boxed{a_1^{(1)} \quad a_2^{(1)} \quad a_i^{(1)} \quad a_n^{(1)}} \\ b^{(1)} \in_r \mathbb{F} & \mathcal{B}^{(1)}(Z) \in_r \mathcal{P}^{b^{(1)}, t} & \boxed{b_1^{(1)} \quad b_2^{(1)} \quad b_i^{(1)} \quad b_n^{(1)}} \\ c^{(1)} \in_r \mathbb{F} & \mathcal{C}^{(1)}(Z) \in_r \mathcal{P}^{c^{(1)}, t} & \boxed{c_1^{(1)} \quad c_2^{(1)} \quad c_i^{(1)} \quad c_n^{(1)}} \end{cases}$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$c^{(L)} = a^{(L)} b^{(L)} \begin{cases} a^{(L)} \in_r \mathbb{F} & \mathcal{A}^{(L)}(Z) \in_r \mathcal{P}^{a^{(L)}, t} & \boxed{a_1^{(L)} \quad a_2^{(L)} \quad a_i^{(L)} \quad a_n^{(L)}} \\ b^{(L)} \in_r \mathbb{F} & \mathcal{B}^{(L)}(Z) \in_r \mathcal{P}^{b^{(L)}, t} & \boxed{b_1^{(L)} \quad b_2^{(L)} \quad b_i^{(L)} \quad b_n^{(L)}} \\ c^{(L)} \in_r \mathbb{F} & \mathcal{C}^{(L)}(Z) \in_r \mathcal{P}^{c^{(L)}, t} & \boxed{c_1^{(L)} \quad c_2^{(L)} \quad c_i^{(L)} \quad c_n^{(L)}} \end{cases}$$

Ashish Choudhury, Arpita Patra : An Efficient Framework for Unconditionally Secure Multiparty Computation. IEEE Trans. Inf. Theory 63(1): 428-468 (2017)

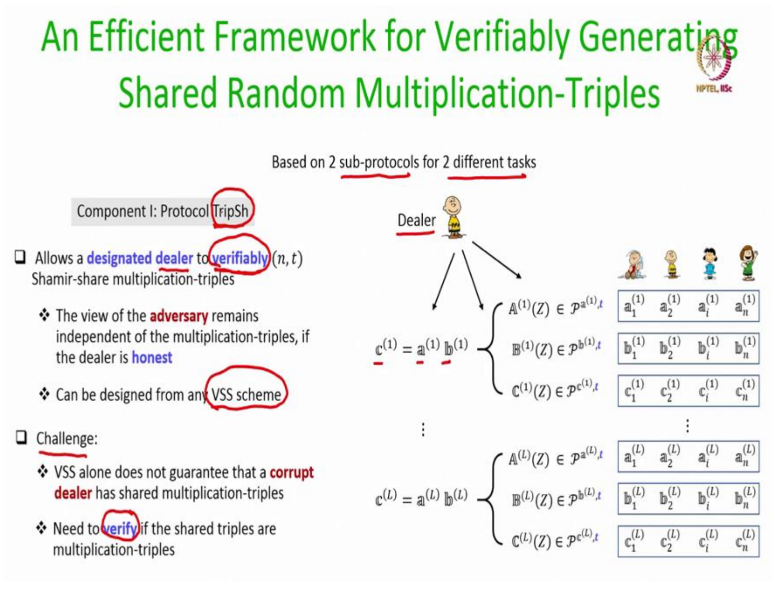
NPTEL, IISc

So, what exactly is the goal for the pre processing phase? The goal here is to verifiably generate secret sharing of random multiplication triplets, say L number of random multiplication triplets, where the triplets are random in the sense that a and b components are random and the c component is the product of the a and b component. And we want that no party should know the exact value of the triplet.

But rather the triplet should be secret shared and corrupt parties should have at most t shares for each of these multiplication triplets which are independent of the underlying multiplication triplets. So, pictorially this is what we want. Corresponding to the first triplet, we require t degree polynomials A1, B1, C1 with each party pi holding the ith point on these polynomials. And like that for the Lth triplet, we want a triplet of polynomials AL, BL and CL.

And each party having the ith point on these polynomials. And all these polynomials should be t degree polynomials. And as I said, for getting these multiplication triplets there are several methods, but I will be discussing a very efficient framework taken from this paper.

(Refer Slide Time: 03:05)



An Efficient Framework for Verifiably Generating Shared Random Multiplication-Triples

Based on 2 sub-protocols for 2 different tasks

Component I: Protocol TripSh

- Allows a **designated dealer** to **verifiably** (n, t) Shamir-share multiplication-triples
  - The view of the **adversary** remains independent of the multiplication-triples, if the dealer is **honest**
  - Can be designed from any VSS scheme
- Challenge:
  - VSS alone does not guarantee that a **corrupt dealer** has shared multiplication-triples
  - Need to **verify** if the shared triples are multiplication-triples

Dealer
$\mathbb{c}^{(1)} = \mathbb{a}^{(1)}\,\mathbb{b}^{(1)}$
$\mathbb{A}^{(1)}(Z) \in \mathcal{P}^{\mathbb{a}^{(1)},t}$
$\mathbb{B}^{(1)}(Z) \in \mathcal{P}^{\mathbb{b}^{(1)},t}$
$\mathbb{C}^{(1)}(Z) \in \mathcal{P}^{\mathbb{c}^{(1)},t}$
$\mathbb{a}_1^{(1)}$ $\mathbb{a}_2^{(1)}$ $\mathbb{a}_i^{(1)}$ $\mathbb{a}_n^{(1)}$
$\mathbb{b}_1^{(1)}$ $\mathbb{b}_2^{(1)}$ $\mathbb{b}_i^{(1)}$ $\mathbb{b}_n^{(1)}$
$\mathbb{c}_1^{(1)}$ $\mathbb{c}_2^{(1)}$ $\mathbb{c}_i^{(1)}$ $\mathbb{c}_n^{(1)}$
$\mathbb{c}^{(L)} = \mathbb{a}^{(L)}\,\mathbb{b}^{(L)}$
$\mathbb{A}^{(L)}(Z) \in \mathcal{P}^{\mathbb{a}^{(L)},t}$
$\mathbb{B}^{(L)}(Z) \in \mathcal{P}^{\mathbb{b}^{(L)},t}$
$\mathbb{C}^{(L)}(Z) \in \mathcal{P}^{\mathbb{c}^{(L)},t}$
$\mathbb{a}_1^{(L)}$ $\mathbb{a}_2^{(L)}$ $\mathbb{a}_i^{(L)}$ $\mathbb{a}_n^{(L)}$
$\mathbb{b}_1^{(L)}$ $\mathbb{b}_2^{(L)}$ $\mathbb{b}_i^{(L)}$ $\mathbb{b}_n^{(L)}$
$\mathbb{c}_1^{(L)}$ $\mathbb{c}_2^{(L)}$ $\mathbb{c}_i^{(L)}$ $\mathbb{c}_n^{(L)}$

So, the framework is based on 2 sub protocols for 2 different tasks. So, I will first explain the 2 sub protocols, what exactly are their requirements, properties. And then we will see that assuming those 2 sub protocols are available, how exactly we can stitch those 2 sub protocols and generate random secret sharing of random multiplication triplets.

The first component is a triple sharing protocol denoted by TripSh. And this is a special type of VSS protocol. Namely there will be a designated dealer which can be any party out of the n parties. And this designated dealer will have some multiplication triplets. So, that will be the input for the dealer.

So, say for instance dealer has l number of multiplication triplets. And this protocol allows the dealer to generate some secret sharing of those triplets in a verifiable fashion such that if the dealer is honest, then the view of the adversary, namely the shares of the adversary, they are distributed randomly over the field and are independent of the dealer's triplets. And the verifiability here ensures that even if the dealer is potentially corrupt, still it will be ensured that it has not shared arbitrary triplets, but rather multiplication triplets.

Because if the dealer is corrupt, then what it can do is it can simply secret share triplets which are not multiplication triplets, namely the c component is not the product of the a and b components. But that is not going to happen here. Because this triple sharing protocol guarantees there will be a verifiability mechanism where the parties will be able to verify whether the dealer has shared triplets which are multiplication triplets.

And then the verification process will also ensure that if the dealer is honest, then during the verification process no information about dealers triplets is revealed. Otherwise the verifiability is very simple; you just reconstruct publicly whatever triples dealer has shared and then check whether they are multiplication triples or not. So, similar to the case of VSS we want here dual properties. We want privacy for the honest dealer, namely dealer's triplets should remain private.

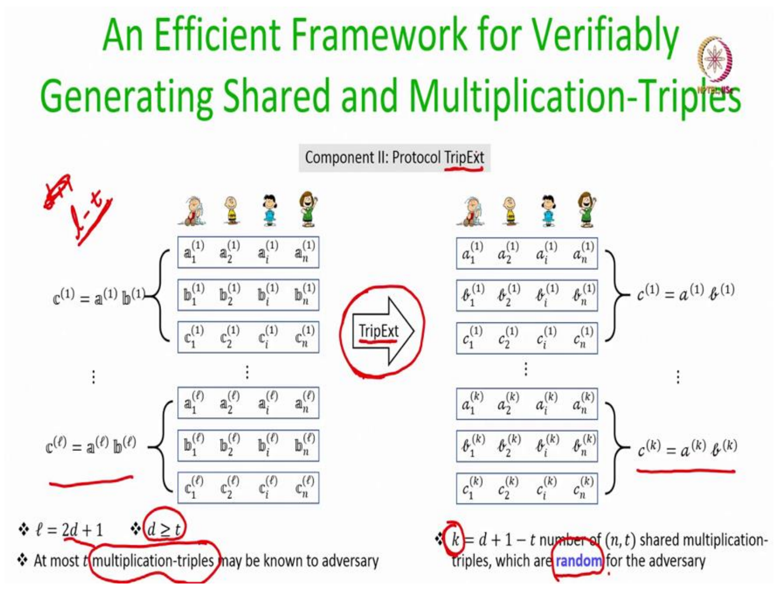
And at the same time we want verifiability in to guarantee that if the dealer has behaved maliciously then it should be caught. Namely if the dealer has shared triplets which are not multiplication triplets then it should be publicly identified. Interestingly this special type of sharing protocol namely this triple sharing protocol can be designed from any polynomial based VSS scheme; however, the challenges are that VSS alone does not guarantee that the corrupt dealer has shared multiplication triplets.

So, if we use a polynomial based VSS and ask the dealer to share the a components b components and c components.

Then it will be ensured that a potentially corrupt dealer has shared its triplets using t degree polynomials, that is fine. And if the dealer is honest then the privacy is maintained, but the strong commitment property of VSS does not guarantee that the a,b,c values which have been secret shared by a potentially corrupted dealer satisfy the multiplicative property. Namely it is not necessary that the c component is the product of a and b component.

So, that is a major challenge, to design this triple sharing protocol on top of VSS we need a verification mechanism. So, VSS itself has a verification mechanism to verify whether the values shared by the dealer are shared as per the t degree polynomial. But on top of that we also need to verify that even the values which are secret shared satisfy the multiplicative relationship and that too without disclosing anything about those values if the dealer is honest. So, that is the main challenge. But for the moment assume we have such a triple sharing protocol, that is the first component of the framework.

(Refer Slide Time: 07:58)


$\ell - t$
$\mathbb{c}^{(1)} = \mathbb{a}^{(1)}\,\mathbb{b}^{(1)}$
$\mathbb{c}^{(\ell)} = \mathbb{a}^{(\ell)}\,\mathbb{b}^{(\ell)}$
TripExt
$c^{(1)} = a^{(1)}\,b^{(1)}$
$c^{(k)} = a^{(k)}\,b^{(k)}$

An Efficient Framework for Verifiably Generating Shared and Multiplication-Triples

Component II: Protocol TripExt

- $\ell = 2d + 1$ $d \geq t$
- At most t multiplication-triples may be known to adversary
- $k = d + 1 - t$ number of (n, t) shared multiplication-triples, which are random for the adversary

The second component is an interesting protocol which is the triple extraction protocol denoted by Trip x Ext and here the input will be the following. Input here will be a bunch of secret shared multiplication triples; that means, there is a bunch of multiplication triples which have been secret shared by different parties.

And now it will be guaranteed here that these triples are multiplication triples. How? We will come to that point later. So, it will be guaranteed that all the triplets here are multiplication triplets and they are secret shared among the parties.

So, we have L number of such multiplication triplets where L = 2d +1 and d will be at least t. And among these L multiplication triplets, the value of at most t triplets will be known to the. And that means, at least L minus t triplets they are random from the viewpoint of the adversary.

That means, for the remaining L minus t multiplication triplets, adversary will just have t shares. So, adversary have shares for all the L triplets plus it may know the exact value of up to t triplets, for the remaining L minus t triplets, it will have only t shares.

However it will not be known which t multiplication triplets are known to the adversary and which L minus t triplets are not known to the adversary. So, that is the crucial part here ok. So, among these bunch of L multiplication triplets, which multiplication triplets are completely known to the adversary and which are not known to the adversary, that information is not public, that is not known to anyone, of course, it is known to adversary; adversary knows which t triplets it knows and which L minus t triplets it does not know.

But the honest parties, they will have absolutely no idea which t triplets are known to the adversary and which L minus t triplets are not known to the adversary, that is the input for this second protocol, triple extraction protocol. Now, as the name triple extraction suggests, what we want to do here, we want a protocol here which allows the parties to generate k number of secret shared multiplication triplets all of which are guaranteed to be unknown for the adversary, random for the adversary ok.

So, let me again stress, among the input multiplication triplets, there are up to L minus t multiplication triplets, which are not known to the adversary. But we do not know which exact L minus t multiplication triplets are not known to the adversary. Somehow we want to do some operation on the input multiplication triplets and extract out k number of secret shared multiplication triplets which are guaranteed to be unknown for the adversary random for the adversary.

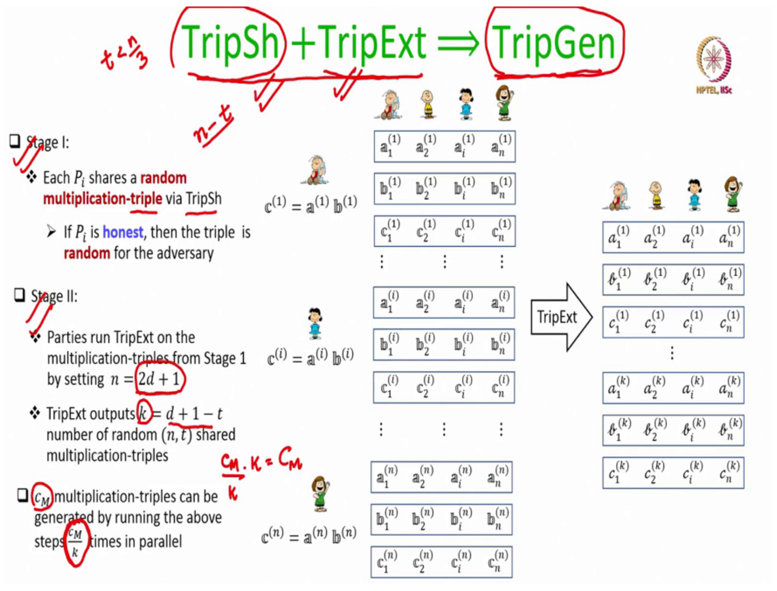
That means, for each of those k triplets, adversary will have only t shares which are randomly distributed. And does not leak anything about the exact value of those triplets ok. And why it is called triple extraction? Because we are given a bunch of input multiplication triplets, some of which are random for the adversary and some of which are not random for the adversary, they are known to the adversary.

Out of this collection of random and non random multiplication triplets we are extracting out a bunch of random multiplication triplets ok.

So, this triple extraction protocol is going to be non trivial. For instance I cannot simply take the first k input triplets as my output triplets, because it could be the case that among the first k input multiplication triplets, some of them are known to the adversary because adversary knows t of them.

So, without even knowing which t input multiplication triplets are known to the adversary, the goal of this triple extraction protocol is to extract out k number of multiplication triplets which are definitely random for that. So, again for the moment we will assume that we have such a triple extraction procedure, later on we will see an instantiation of the triple extraction procedure.

(Refer Slide Time: 13:16)



TripSh + TripExt $\Rightarrow$ TripGen

$t < n/3$ $\quad$ $n-t$

- Stage I:
  - Each P_i shares a **random multiplication-triple** via TripSh
    - If P_i is **honest**, then the triple is **random** for the adversary
- Stage II:
  - Parties run TripExt on the multiplication-triples from Stage 1 by setting $n = 2d + 1$
  - TripExt outputs $k = d + 1 - t$ number of random (n, t) shared multiplication-triples
- c_M multiplication-triples can be generated by running the above steps $\frac{c_M}{k}$ times in parallel

$\frac{c_M}{k} \cdot k = c_M$

$\mathbb{c}^{(1)} = \mathbb{a}^{(1)}\,\mathbb{b}^{(1)}$

$\mathbb{c}^{(i)} = \mathbb{a}^{(i)}\,\mathbb{b}^{(i)}$

$\mathbb{c}^{(n)} = \mathbb{a}^{(n)}\,\mathbb{b}^{(n)}$

Now, the framework does the following. Assuming we have an instantiation of the triple sharing protocol and an instantiation of the triple extraction protocol, how we can stitch them together to get the triple generation protocol? Namely the protocol for the pre processing phase. So, this triple generation means this is the protocol which allows the parties to generate random secret sharing of random multiplication triplets.

So, this will be a two stage process. Each stage will have a different purpose. In the first stage, each party will be asked to act as a dealer and secret share a random multiplication triplet by acting as a dealer and invoking an instance of the verifiable triple sharing protocol.

So, we will see that by combining the stage 1 and stage 2, how many triplets we can generate? And then if the goal is to generate cM number of random multiplication triplets where cM is the number of multiplication gates in the circuit, which the parties want to compute. Then we have to see how many times this stitching has to be done.

So, in the stage 1, each party will be acting as a dealer and secret share random multiplication triplet by acting as a dealer by playing the role of the dealer ok. So, P1 for instance will pick a multiplication triplet, which is random and it will act as a dealer and secret share it. In parallel the ith party will also do the same and in parallel the nth party also will do the same. Now, what exactly this stage 1 guarantees? It guarantees that if the ith party is honest, then in its instance of the triple sharing protocol whatever triplet it has shared it is a multiplication triple and that is random for the adversary. Moreover even if Pi is corrupt, the verifiability of the triple sharing protocol guarantees that it has shared a multiplication triple only, it is not just a triple it is a multiplication triple. But in that case since the party P i is corrupt it may not share a random triplet it could share a multiplication triplet whose distribution is not random.

But if party Pi is honest, then in its instance of the triple sharing protocol, it is bound to share a random multiplication triplet about which the adversary will have absolutely no idea. So, since we are working in the setting t less than n over 3 and there up there could be up to t corrupt parties, we know that there are at least n minus t multiplication triplets at the end of stage one which are random for the adversary.

However since the exact identity of the honest parties is not known we do not know which are those n minus t random multiplication triples. Now, at the end of the stage one we have a set of n multiplication triples, n minus t of them are random, t of them may not be random we do not know which are random and which are not random.

So, what we can do is, in stage two we can run the triple extraction procedure on the shared multiplication triples which we have at the end of stage one by setting n to be 2 d plus 1.

So, for simplicity we assume that n is of the form 2 d plus 1. So, if we apply the triple extraction procedure then the triple extraction procedure will guarantee that at the end of the stage 2 they are d plus 1 minus t number of random multiplication triples none of which is known to the adversary and all those multiplication triplets are randomly secret shared through t degree sharing polynomials.

So, now by doing this stage 1 and stage 2 we are able to get k number of random multiplication triples. If the goal of the pre processing phase is to get cM number of random secret shared multiplication triples then the above procedure can be executed in parallel cM over k number of times. Through one instance of stage 1 and stage 2 we are able to get k number of triples, hence by running cM over k number of instances of stage 1 and stage 2 in parallel, we will be able to manage cM over k into k, which is equal to cM number of random secret shared multiplication triples.

So, that is the idea behind the framework. So, now, everything boils down to how exactly we get this triple sharing protocol and how exactly we get this triple extraction protocol.

And from the description it is easy to see that this framework is very generic you give me any triple sharing instantiation and any triple extraction which have the properties as mentioned earlier. Then by stitching them together in this fashion we will be able to get cM number of random secret shared multiplication triples. So, that is the good part about this frame work. So, in the next lecture we will see how do we go about instantiating this triple sharing protocol and the triple extraction protocol.

Thank you.