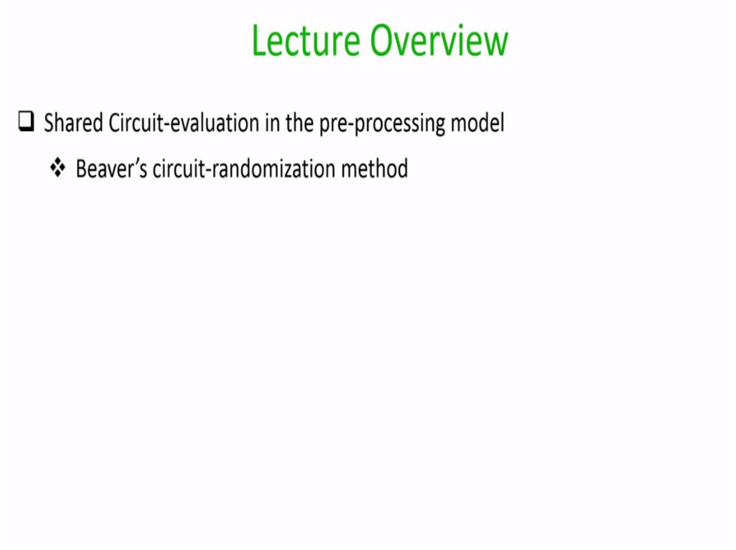
Secure Computation: Part 1

Prof. Ashish Choudhury
Department of Computer Science
Indian Institute of Science – Bengaluru

Lecture – 27
Shared Circuit-Evaluation in the Pre-Processing Model

Hello everyone. Welcome to this lecture.

(Refer Slide Time: 00:31)

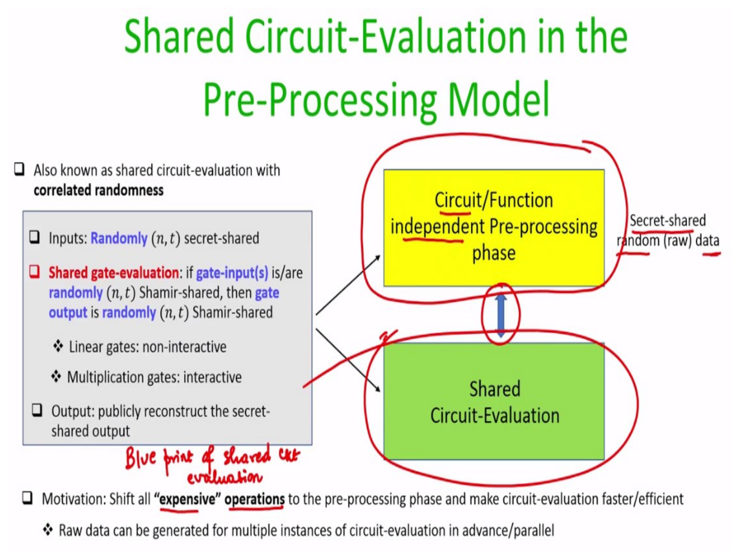


Lecture Overview

- ❑ Shared Circuit-evaluation in the pre-processing model
  - ❖ Beaver's circuit-randomization method

So in this lecture we will see how we can perform shared circuit evaluation and what we call as the pre-processing model which is now the de facto way of performing shared circuit evaluation in generic MPC protocols and under that we will see a very popular approach which we call as the Beaver's circuit randomization method.

(Refer Slide Time: 00:58)



Shared Circuit-Evaluation in the Pre-Processing Model

- ❑ Also known as shared circuit-evaluation with **correlated randomness**

- ❑ Inputs: Randomly (n,t) secret-shared
- ❑ Shared gate-evaluation: if gate-input(s) is/are randomly (n,t) Shamir-shared, then gate output is randomly (n,t) Shamir-shared
  - ❖ Linear gates: non-interactive
  - ❖ Multiplication gates: interactive
- ❑ Output: publicly reconstruct the secret-shared output

Circuit/Function independent Pre-processing phase
Secret-shared random (raw) data
Shared Circuit-Evaluation
Blue print of shared ckt evaluation

- ❑ Motivation: Shift all **"expensive" operations** to the pre-processing phase and make circuit-evaluation faster/efficient
  - ❖ Raw data can be generated for multiple instances of circuit-evaluation in advance/parallel

So, what exactly is the shared circuit evaluation in the pre-processing model? This process is also called as shared circuit evaluation with co-related randomness. So, this was the blue print of shared circuit evaluation. You will secret share the inputs and then evaluate each intermediate gate in a secret shared fashion and then finally go and reconstruct the function output.

So, in the pre-processing model the idea is that we somehow divide this whole process of shared circuit evaluation into two independent task. Stage one or task one is the pre-processing stage where we produce some kind of secret shared data of course it will be a random data, but it would not be a completely random data it will have some kind of relation among itself.

But the point is that random data the so called random data that we are generating here that will be independent of the actual function, of the actual circuit which parties would like to evaluate or compute and also it will be independent of the inputs of the parties for the function or for the circuit that is important. So, it is a circuit and function independent pre-processing phase.

Again this will be done by the parties themselves and the second stage is the actual shared circuit evaluation and of course there will be interaction among these two stages, interaction among these two stages in the sense that whatever data has been generated in the function independent pre-processing phase that will be later utilized by this second stage. The main motivation behind splitting this shared circuit evaluation into this two independent processes, two independent stages is the following.

The idea will be that we would like to shift all the expensive operation expensive means which require enormous amount of communication, computation to this pre-processing phase with the hope that the circuit evaluation becomes faster and why this approach is really interesting because when we were doing the pre processing since it is independent the data which we are generating there it is independent of the actual function or the actual computation which the parties would like to perform.

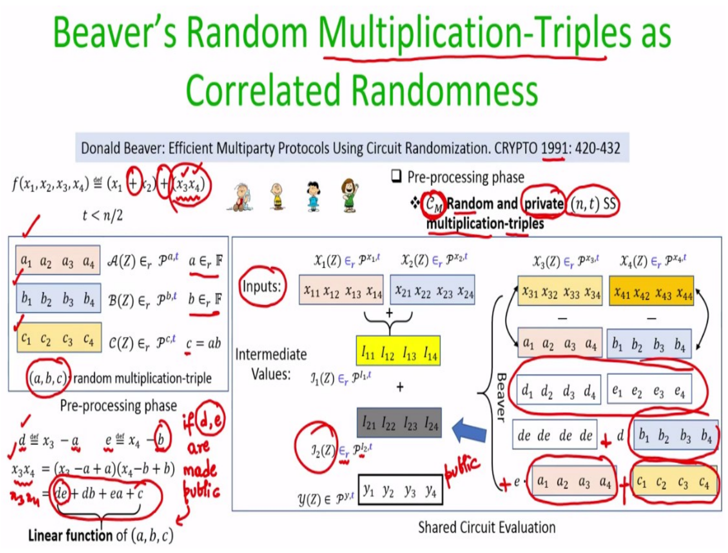
This raw data can be generated in abundance for multiple instances of shared circuit evaluation that means if the same set of parties are interested to compute the same function over multiple inputs. So, that means the same circuit has to be evaluated multiple times, but over possible different inputs then this pre-processing would allow us to generate whatever raw data we require for those many instances of shared circuit evaluation we can generate the raw data for all those shared circuit evaluation in advance and that too in parallel.

We do not have to depend that okay let the first instance of shared circuit evaluation get over and then we start pre-processing phase for the next instance of shared circuit evaluation and so on. So, it is like saying the following if you want to evaluate the same function say 100 times or 1,000 times generate whatever raw data you require for 100 evaluations or in fact more than 100 evaluations and keep them in a bank in the sense in the memory of the parties in a secret shared fashion.

And then as and when the new instances of shared circuit evaluation starts you pull back the data that you have generated in the pre-processing phase and utilize it and hope that your actual shared circuit evaluation is very, very fast and this makes it a very practical approach when we come to the shared circuit evaluation because typically if we consider the real world applications of MPC it is very often the case that the same set of parties are involved to compute the same function over different possible inputs.

So, if we can somehow ensure that at the time of the actual shared circuit evaluation the process is going to be very, very fast based on the assumption that we have already generated pre-processing data then this is really a very practically motivated approach. So, now this correlated randomness that the parties are going to generate in the pre-processing phase it can take multiple form it is not the case that there is only one form of pre-processing data which can be later used for the shared circuit evaluation.

(Refer Slide Time: 06:05)



Beaver's Random Multiplication-Triples as Correlated Randomness
Donald Beaver: Efficient Multiparty Protocols Using Circuit Randomization. CRYPTO 1991: 420-432
$f(x_1, x_2, x_3, x_4) \overset{\text{def}}{=} (x_1 + x_2) + (x_3 x_4)$
$t < n/2$
Pre-processing phase
c_M Random and private (n, t) SS multiplication-triples
$a_1\ a_2\ a_3\ a_4$ $\mathcal{A}(Z) \in_r \mathcal{P}^{a,t}$ $a \in_r \mathbb{F}$
$b_1\ b_2\ b_3\ b_4$ $\mathcal{B}(Z) \in_r \mathcal{P}^{b,t}$ $b \in_r \mathbb{F}$
$c_1\ c_2\ c_3\ c_4$ $\mathcal{C}(Z) \in_r \mathcal{P}^{c,t}$ $c = ab$
(a, b, c) random multiplication-triple
Pre-processing phase
$d \overset{\text{def}}{=} x_3 - a$ $e \overset{\text{def}}{=} x_4 - b$
if (d, e) are made public
$x_3x_4 = (x_3 - a + a)(x_4 - b + b)$
$= de + db + ea + c$
Linear function of (a, b, c)
Inputs:
Intermediate Values:
Beaver
public
Shared Circuit Evaluation

And indeed in the literature there are several form of correlated randomness which have been proposed each of these methods have its own trade off pros and cons. We are going to discuss in this lecture one of the most popular form of correlated randomness which are also called as multiplication triples or Beaver's multiplication triples and this is based on a very clever idea proposed long back in 1991 by Beaver.

Again for the sake of demonstration I will take this example circuit and remember we are doing the pre-processing to make the evaluation of the multiplication gates faster in the actual shared circuit evaluation because evaluating the linear gates during the shared circuit evaluation does not require any interaction among the parties. So, we are doing this whole process to make the evaluation of the multiplication gates faster in the online phase or in the actual shared circuit evaluation.

So, what is the form of pre-processing data that is generated in this method. The pre-processing phase generates what we call as random multiplication triples in n, t secret shared fashion, Shamir secret shared fashion or it could be as per any linear secret sharing scheme as per any n, t linear secret sharing scheme, but for the purpose of demonstration we will focus on Shamir secret sharing.

So, it generates a random and shared multiplication triples and they will be private in the sense the actual value of the multiplication triples will not be known to any party, but the parties will just have their respective shares of those triples and if there are C M number of multiplication gates in the circuit then basically in the pre-processing phase we should have generate C M or more number of multiplication triples.

Then only we can utilize them later for the shared secret evaluation because later when we are going to perform the shared circuit evaluation we require that for each multiplication gate there should be one independent multiplication triple which have been generated in the pre-processing phase. So that is why if there are C M number of multiplication triples in the circuit we should have C M number of multiplication triple generated in the pre-processing phase.

Of course, you can generate more than C M as well you can generate million time C M number of random triples and so on. So, in this specific example there is only one multiplication gate so we will assume that in the pre-processing phase the parties somehow have generated the following data structure. How they are going to generate this data structure that will be the focus of the rest of the lecture.

But I am going to first show that assuming that such a data structure has been generated in the pre-processing phase how the evaluation of the multiplication gate simplified. So, in the pre-processing phase somehow the parties would have generated secret shares of the secrets a, b, c where a, b, c are going to be random field elements. Of course c is not a random element a and b are the random field elements and c will be the product of those a and b.

It will be known that c is the product of a and b, but the exact value of a, exact value of b and exact of c will not be known, but rather the parties will hold shares of a, shares of b and shares of c that is a data structure we are assuming that have been already generated in the pre-processing phase. So, for example your a, b, c could be 0, 0, 0 your a, b, c could be 1, 0, 0, 0, 1, 0 or it could be 1, 1, 1 or it could be 2, 2, 4.

Depending upon what is the value of a and what is the value of b and if I have more than one multiplication gate the parties will have such data structure generated C M number of times independent of each other. Now, assuming such pre-processing has been already done here is how the actual shared circuit evaluation will be performed. Inputs of the function they will be secret shared as usual.

The linear gate it will be evaluated locally by parties just locally adding their respective shares of the inputs of the corresponding linear gate. Now we have this multiplication gate. So, earlier approach was the parties do a GRR degree reduction and then from the shares of x 3 and shares of x 4 they obtain their respective shares of x 3 times x 4, but we are not going to do that because that requires communication of n square field elements.

What we are going to instead do is the following. Our goal is to compute secret sharing of x 3 times x 4 and let d and e be the one time pad encryption of x 3 and x 4 with a and b serving as the pads. Right now no one knows the value of d and e. Our goal is to compute x 3 times x 4 so x 3 times x 4 can be rewritten like this and now if I expand further I can write down x 3 times x 4 to be this expression.

And now if you see closely here my x 3 times x 4 turns out to be a linear function of a, b, c why linear function of a, b, c? Assuming that I make d and e public somehow if d, e are made public then x 3 times x 4 is a linear function of a, b and c with d, e serving as the linear combiners because you have this d times e which is anyhow public. You have d times b, you have e times a plus you have one time c so it is a linear function of a, b, c.

And we know that if the inputs of a linear function are available in a secret shared fashion then the output of that linear function can also be made available in a secret shared fashion that means based on this observation we notice that b, a and c are anyhow available in a secret shared fashion. If somehow we can make d and e public then by applying the same linear combination the parties will end up getting their respective shares of x 3 times x 4.

And that is what we want to do when I say I want to evaluate this multiplication gate that means in this whole thing that means this evaluating this multiplication gates x 3 times x 4 in a secret shared fashion reduces to evaluating this linear function in a secret shared fashion. Given that d and e are made public. So, what we have to do is we have to just make d and e public.

And after that the parties can compute the shares of x 3 times x 4 locally because remember evaluating linear gates does not require any interaction if the inputs are secret shared. How do we make d and e public? Well, d is the difference of x 3 and a. X 3 right now is available in a secret shared fashion a is also available in a secret shared fashion. So, parties can first compute the secret sharing of x 3 – x 1 by locally subtracting the shares of a from the shares of x 3.

And then they will obtain a secret sharing of d right now no communication has happened in the same way parties can locally compute a secret sharing of e and now they can exchange shares of d and e amongst themselves and make d and e public that is all and making d and e public does not require too much of communication we can follow order n communication complexity reconstruction protocol and reconstructing two values d and e.

Namely the one time pad encryption of x 3 and x 4 will require order n communication and you can see now no GRR degree reduction is coming into picture just you have to reconstruct two secret shared values namely d and e and now once d and e are public we have to just compute this linear function in secret shared fashion which we know how to do d times e is a public constant.

So, parties take a default share of d times e as the share of d times e and then they have their respective share of b which they multiply locally with d, they have their respective shares of a which they locally multiply with e and they have their respective shares of c and now if they add all of them together they obtain their respective shares of output of this multiplication gate in a secret shared fashion.

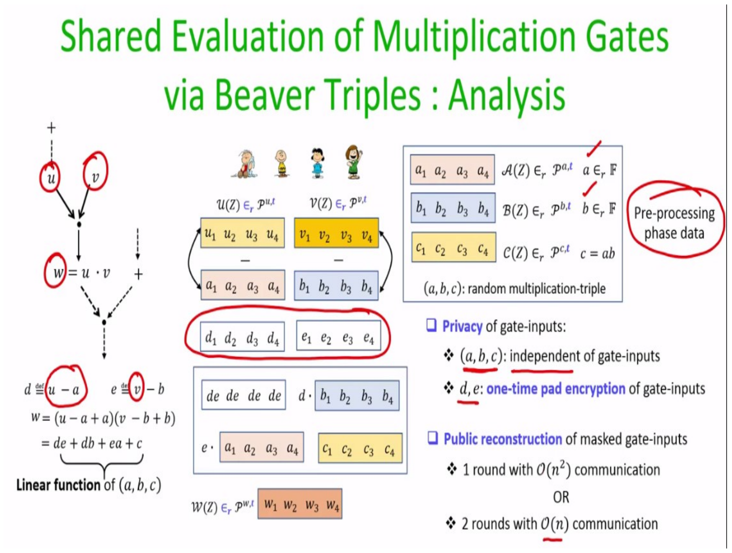
And now it is easy to see that these shares lie on a random t degree polynomial. Why random t degree polynomial? Because in the pre-processing phase the b shares lie on a random polynomial the a shares lie on a random polynomial and even the c shares lie on a random polynomial of degree t each and if you add or you take linear combination of several random t degree polynomial the overall polynomial also is going to be a random t degree polynomial except the constant term would not be random.

But rest of the coefficients will be random and that is what we wanted to do and now once you evaluate this gate after that you have a addition gate which can be computed locally and then the parties can make it public the y shares and that is how you evaluate the multiplication gate that means evaluating multiplication gate as per this method now boils down to publically reconstructing two values.

If you have another multiplication gate the parties will require such random shared multiplication triple and then they can follow the same process here. Now in this whole process the privacy of x 3 and x 4 is maintained. Why even though x 3 – a is made public since a was private and randomly chosen, b was private and randomly chosen hence learning d and e does not reveal anything about x 3 and x 4 because it could be any x 3 and any a whose difference would have produced this t.

That means for every candidate x 3 there is a corresponding candidate a such that the difference of x 3 and a would have produce the d and similar arguments for e, but since a and b are picked randomly it could be any x 3 along with the corresponding a which would have produced this d and this d. Hence the privacy of x 3 and x 4 is maintained.

(Refer Slide Time: 17:47)



Shared Evaluation of Multiplication Gates via Beaver Triples : Analysis

- Privacy of gate-inputs:
  - (a,b,c): independent of gate-inputs
  - d,e: one-time pad encryption of gate-inputs
- Public reconstruction of masked gate-inputs
  - 1 round with $\mathcal{O}(n^2)$ communication

    OR
  - 2 rounds with $\mathcal{O}(n)$ communication

$w = u \cdot v$
$d \overset{\text{def}}{=} u - a$
$e \overset{\text{def}}{=} v - b$
$w = (u - a + a)(v - b + b)$
$= \underbrace{de + db + ea + c}_{\textbf{Linear function} \text{ of } (a,b,c)}$
$\mathcal{U}(Z) \in_r \mathcal{P}^{u,t}$
$\mathcal{V}(Z) \in_r \mathcal{P}^{v,t}$
$u_1\ u_2\ u_3\ u_4$
$v_1\ v_2\ v_3\ v_4$
$a_1\ a_2\ a_3\ a_4$
$b_1\ b_2\ b_3\ b_4$
$d_1\ d_2\ d_3\ d_4$
$e_1\ e_2\ e_3\ e_4$
$de\ de\ de\ de$
$d \cdot b_1\ b_2\ b_3\ b_4$
$e \cdot a_1\ a_2\ a_3\ a_4$
$c_1\ c_2\ c_3\ c_4$
$\mathcal{W}(Z) \in_r \mathcal{P}^{w,t}$ $w_1\ w_2\ w_3\ w_4$
$\mathcal{A}(Z) \in_r \mathcal{P}^{a,t}$ $a \in_r \mathbb{F}$
$\mathcal{B}(Z) \in_r \mathcal{P}^{b,t}$ $b \in_r \mathbb{F}$
$\mathcal{C}(Z) \in_r \mathcal{P}^{c,t}$ $c = ab$
(a,b,c): random multiplication-triple
Pre-processing phase data

So, now let us do the full analysis of the shared circuit evaluation assuming we have such a pre-processing phase and I stress here that we have not seen the actual pre-processing phase this we will see very soon. So, imagine you have a multiplication gate somewhere in the circuit corresponding to that the parties will have this pre-processing data already generated the goal of the parties will be the following if they have their respective shares of u their respective shares of v lying on t degree random polynomial.

They would like to have their respective shares of w lying on a t degree polynomial. For that they basically have to compute the one time pad encryption of u and v as per the secret shared a and secret shared b and make it public and then they compute the linear combination. As I said the privacy of the gate inputs is preserved because a, b, c they are picked independent of the gate inputs.

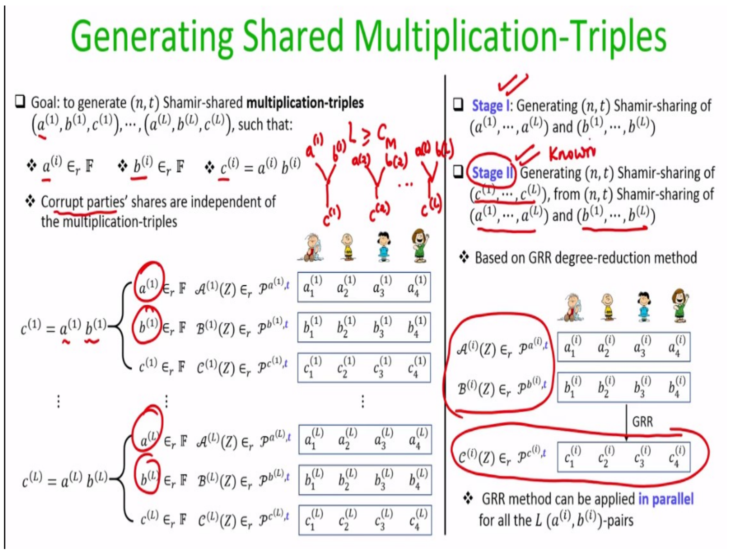
Remember when I discuss the requirement of the pre-processing phase I said that the pre-processing data have no dependency on the values of u, v and w and so on. So, a, b, c had no dependency whatsoever on u, v and w and d and e are here acting as one time pad encryption of the gate inputs. So, that tells you that if you have multiple multiplication gates in the circuit you require independent a, b, c you cannot reuse the same multiplication triple for all the multiplication gates in the circuit because if you make public the OTP encryptions of all the multiplication gate inputs using the same a and same b respectively.

Then we know that if the same key is used for encrypting multiple values then we no longer can argue perfect privacy, but when I say a, b, c are picked randomly and independently and there are several such a, b, c which are independent of each other that does not mean that the values of a, b and hence c they are not allowed to be repeated no they are picked independently that is what I mean here.

When I say independent random multiplication triples. So, how much communication is required for evaluating one multiplication gate assuming we have this pre-processing data so depending upon what kind of reconstruction protocol you use. So, the whole cost for evaluating a multiplication gate is basically reconstructing two secret shared values that is the only overhead for evaluating a multiplication gate.

So, now it is up to you if you found to follow this order n communication complexity reconstruction protocol then you get a saving here and this is now better than the previous method of applying the GRR degree reduction method to evaluate this multiplication gate because that has an overhead of n square field element, but we can now reduce the overhead to order n per multiplication gate if we follow this process.

(Refer Slide Time: 20:58)



Generating Shared Multiplication-Triples

- Goal: to 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

$L \geq C_M$

$c^{(1)} = a^{(1)}\, b^{(1)}$: $a^{(1)} \in_r \mathbb{F}$, $\mathcal{A}^{(1)}(Z) \in_r \mathcal{P}^{a^{(1)},t}$: $a_1^{(1)}$ $a_2^{(1)}$ $a_3^{(1)}$ $a_4^{(1)}$; $b^{(1)} \in_r \mathbb{F}$, $\mathcal{B}^{(1)}(Z) \in_r \mathcal{P}^{b^{(1)},t}$: $b_1^{(1)}$ $b_2^{(1)}$ $b_3^{(1)}$ $b_4^{(1)}$; $c^{(1)} \in_r \mathbb{F}$, $\mathcal{C}^{(1)}(Z) \in_r \mathcal{P}^{c^{(1)},t}$: $c_1^{(1)}$ $c_2^{(1)}$ $c_3^{(1)}$ $c_4^{(1)}$

⋮

$c^{(L)} = a^{(L)}\, b^{(L)}$: $a^{(L)} \in_r \mathbb{F}$, $\mathcal{A}^{(L)}(Z) \in_r \mathcal{P}^{a^{(L)},t}$: $a_1^{(L)}$ $a_2^{(L)}$ $a_3^{(L)}$ $a_4^{(L)}$; $b^{(L)} \in_r \mathbb{F}$, $\mathcal{B}^{(L)}(Z) \in_r \mathcal{P}^{b^{(L)},t}$: $b_1^{(L)}$ $b_2^{(L)}$ $b_3^{(L)}$ $b_4^{(L)}$; $c^{(L)} \in_r \mathbb{F}$, $\mathcal{C}^{(L)}(Z) \in_r \mathcal{P}^{c^{(L)},t}$: $c_1^{(L)}$ $c_2^{(L)}$ $c_3^{(L)}$ $c_4^{(L)}$

- **Stage I**: Generating (n,t) Shamir-sharing of $(a^{(1)},\cdots,a^{(L)})$ and $(b^{(1)},\cdots,b^{(L)})$
- **Stage II** Known: Generating (n,t) Shamir-sharing of $(c^{(1)},\cdots,c^{(L)})$, from (n,t) Shamir-sharing of $(a^{(1)},\cdots,a^{(L)})$ and $(b^{(1)},\cdots,b^{(L)})$
  - Based on GRR degree-reduction method

$\mathcal{A}^{(i)}(Z) \in_r \mathcal{P}^{a^{(i)},t}$: $a_1^{(i)}$ $a_2^{(i)}$ $a_3^{(i)}$ $a_4^{(i)}$

$\mathcal{B}^{(i)}(Z) \in_r \mathcal{P}^{b^{(i)},t}$: $b_1^{(i)}$ $b_2^{(i)}$ $b_3^{(i)}$ $b_4^{(i)}$

GRR

$\mathcal{C}^{(i)}(Z) \in_r \mathcal{P}^{c^{(i)},t}$: $c_1^{(i)}$ $c_2^{(i)}$ $c_3^{(i)}$ $c_4^{(i)}$

- GRR method can be applied **in parallel** for all the L $(a^{(i)},b^{(i)})$-pairs

So, now we have to understand that how exactly we are going to generate that pre-processing data. The whole thing works under the assumption that this pre-processing data is given to you, but how exactly do you generate this pre-processing data, there is no trusted third party who is going to come and produce this pre-processing data for you it is a parties only who have to do computation collectively and generate the pre-processing data.

So, now we will see the pre-processing phase of the protocol. So, the goal is to generate L number of secret shared random multiplication triples where L is some parameter. What do I mean by shared random multiplication triples? So, the a component of the triple, the b component of the triples should be uniformly random and picked independently and the c component should be the product of the corresponding a and b component.

That is what I mean by a, b, c being random multiplication triples and we want that in this whole process of generating the multiplication triples the corrupt parties view or their shares should be independent of the actual value of the underlying a, b, c that means from their view point it could be any a, b, c which have been generated in the pre-processing phase and they are saying just t random shares for those randomly generated a, b, c that is what we want to ensure.

So, pictorially since we want to generate L such multiplication triples here is what we want to generate. We want to generate one triple, a 1, b 1, c 1 where c 1 is the product of a 1 b 1 a 1 being randomly secret shared through a t degree polynomial and parties holding shares of them b 1 being secret shared through a random t degree polynomial c 1 being randomly secret shared through t degree polynomial and so on.

And I stress that it is quite possible that a components of all the multiplication triples are same, b components of all the multiplication triples are same and c component is the corresponding product of a and b that is quite possible. Do not think that a, b, c they have to be the unique values no they have to be independently generated that is all. We only want to ensure that adversary does not learn any information regarding the exact value of a, b, c.

It should just have t shares and those t shares should be independent of the exact value of the a, b, c components of the multiplication triples that is what we want to do and we do this in two stages. Of course here L is a parameter you can fix it L could be million, billion depending upon how many such triples parties can generate in parallel. Of course L has to be greater than equal to C M because if you want to evaluate a circuit which has C M number of multiplication gates.

Then you need to at least generate those many secret shared multiplication triples in advance. So, we generate this database in two stages. In stage 1 we first generate the secret sharing a component and b component. So, we generate all the secret shared a 1, a 2, a L satisfying the requirements and in parallel the secret shared b components. Remember a and b component they are independent of each other.

So, it is like saying the following in stage 1 I generate random secret sharing of 2 L number of random elements from the field that is what we are going to do in stage 1 of the pre-processing phase. We will see how we do that assuming stage 1 is taken care now we have to generate the c component of the each multiplication triple that means given this secret shared a 1, a 2, a L secret shared b 1, b 2, b L we have to now generate secret shared c 1 to c L.

And now for this we use the GRR degree reduction method namely for the Ith triple assuming that the parties have shares of these two random sharing polynomials. We apply the GRR degree reduction method to generate the corresponding C polynomial and their shares. Now you might be wondering that we did all these things to kind of avoid GRR degree reduction in the actual shared circuit evaluation.

But when we are producing the random multiplication triples in the online phase we are again using GRR degree reduction so what is the whole point. The advantage here is the following now all this multiplication triples they can be generated in parallels so a 1 is independent of a 2 b 2 that is independent of a 3 b 3 that is independent of a 4, b 4 and so on. So, now what you can imagine here is the following.

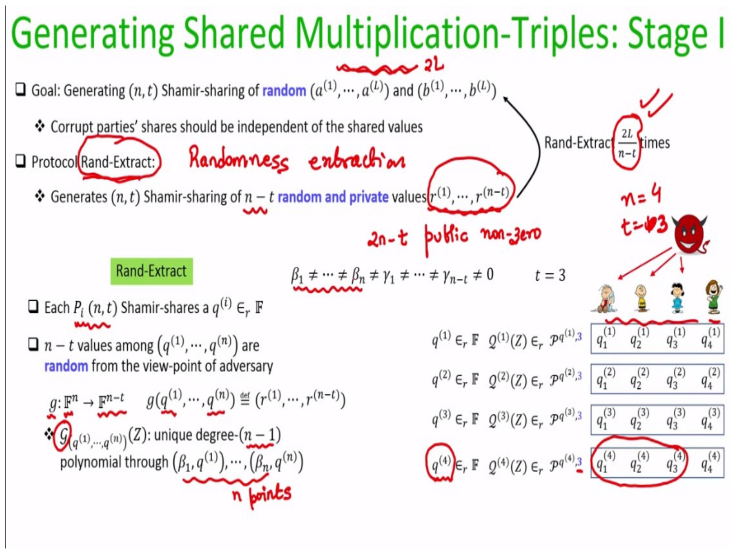
In stage you have a circuit where the multiplicative depth is just 1 why 1 because you can imagine that there is a circuit where you have L number of multiplication gates all of them lying at the same layer which is what you want to evaluate in stage 2 and this means that all the instances of GRR degree reduction can be executed in parallel. This is unlikely the case where we perform shared circuit evaluation based on GRR degree reduction method because in that case depending upon the order in which the multiplication gates are there in the circuit the actual circuit for evaluating the function F.

There might be a dependency that I cannot run the second instance of GRR degree reduction until and unless the earlier instances of GRR degree reductions are over, but if I prepone all the instances of GRR degree reduction like this as I am doing right now proposing right now for the pre-processing phase stage 2 then basically I can run all the instances of GRR degree reduction in parallel and that results in reduction in the number of rounds.

Later on in the next course when we will consider the malicious setting it turns out that actually by pre preponing all the instances of GRR degree reduction to this pre-processing phase we get significant saving not only in terms of rounds, but also in terms of communication complexity as well. So, that means this stage 2 is anyhow known we know how to do or how to compute the c components of the shared multiplication triples.

Assuming that the a components and the b components have been generated. So, our focus now should be how do we generate this secret shared a components and b components.

(Refer Slide Time: 28:16)



Generating Shared Multiplication-Triples: Stage I

2L

- Goal: Generating (n, t) Shamir-sharing of **random** $(a^{(1)}, \cdots, a^{(L)})$ and $(b^{(1)}, \cdots, b^{(L)})$
  - Corrupt parties' shares should be independent of the shared values
- Protocol Rand-Extract: Randomness extraction
  - Generates (n, t) Shamir-sharing of $n - t$ **random and private** values $r^{(1)}, \cdots, r^{(n-t)}$

Rand-Extract $\frac{2L}{n-t}$ times

2n-t public non-zero

n = 4

t = ~~1~~3

Rand-Extract

$\beta_1 \neq \cdots \neq \beta_n \neq \gamma_1 \neq \cdots \neq \gamma_{n-t} \neq 0$ $\quad t = 3$

- Each P_i (n, t) Shamir-shares a $q^{(i)} \in_r \mathbb{F}$
- $n - t$ values among $(q^{(1)}, \cdots, q^{(n)})$ are **random** from the view-point of adversary

$g: \mathbb{F}^n \rightarrow \mathbb{F}^{n-t}$ $\quad g(q^{(1)}, \cdots, q^{(n)}) \overset{\text{def}}{=} (r^{(1)}, \cdots, r^{(n-t)})$

- $\mathcal{G}_{(q^{(1)}, \cdots, q^{(n)})}(Z)$: unique degree-$(n-1)$ polynomial through $(\beta_1, q^{(1)}), \cdots, (\beta_n, q^{(n)})$

n points

$q^{(1)} \in_r \mathbb{F}$ $\quad \mathcal{Q}^{(1)}(Z) \in_r \mathcal{P}^{q^{(1)},3}$ $\quad \boxed{q_1^{(1)} \quad q_2^{(1)} \quad q_3^{(1)} \quad q_4^{(1)}}$

$q^{(2)} \in_r \mathbb{F}$ $\quad \mathcal{Q}^{(2)}(Z) \in_r \mathcal{P}^{q^{(2)},3}$ $\quad \boxed{q_1^{(2)} \quad q_2^{(2)} \quad q_3^{(2)} \quad q_4^{(2)}}$

$q^{(3)} \in_r \mathbb{F}$ $\quad \mathcal{Q}^{(3)}(Z) \in_r \mathcal{P}^{q^{(3)},3}$ $\quad \boxed{q_1^{(3)} \quad q_2^{(3)} \quad q_3^{(3)} \quad q_4^{(3)}}$

$q^{(4)} \in_r \mathbb{F}$ $\quad \mathcal{Q}^{(4)}(Z) \in_r \mathcal{P}^{q^{(4)},3}$ $\quad \boxed{q_1^{(4)} \quad q_2^{(4)} \quad q_3^{(4)} \quad q_4^{(4)}}$

So, our goal is this and for this we will use a protocol call as Rand extract standing for randomness extraction and this protocol does the following. Here the parties have no input, but it will be a randomized protocol and it will generate secret sharing of n – t number of random values which will be private from the view point of the adversary and adversary will have just t shares for each of this random values that is what one instance of random extraction is going to produce as output.

But our goal is to generate 2 L number of random secret shared values. So, how we can use this Rand Extract protocol to generate 2 L number of random secret shared values well we can run this Rand randomness extraction protocol these many number of times because through each instance n – t number of random secret shared values are going to be generated as the outcome.

And if I generate these many number of instances in parallel then overall I will be ending up generating these many number of 2 L number of random secret shared values. So, now let us see one instance of randomness extraction and in this randomness extraction process we will assume that there is a public setup namely the parties known 2 n – t number of public non zero elements from the field which are going to be used as some evaluation points.

Some of these evaluation points could be the same as your alpha components which we are using as part of our secret sharing, but they might be different as well and for the purpose of demonstrating the randomness extraction process I will assume n = 4 and t = 1. Our goal is to generate n – t number of secret shared values which are random and random from the viewpoint of the adversary as well.

So, in this specific case I will consider a process which should allow this four parties to collectively generate secret sharing of one value because n – t is 1 which should not be known to the adversary and every party just having a share of that random value. So, to begin with the parties locally pick respectively a random values from the field and Shamir shares it. So, for instance, P 1 will pick q 1 and secret share it through a random polynomial of degree 3.

P 2 independently and in parallel will pick a random value q 2 and P 3 independently will pick a random value q 3 and P 4 will independently pick a random value q 4 and all of them will be independently secret shared. So now n number of q values have been secret shared because we are n parties so hence n number of q values have been secret shared out of this n number of q shared values t are shared by corrupt parties which are under the control of adversary.

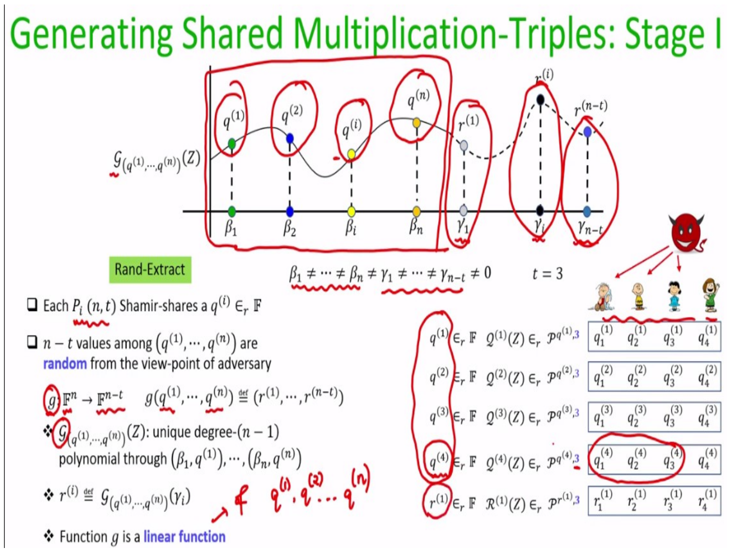
And n – t are shared by the honest parties which are not under the control of the adversary. So, in this specific demonstration the value q 4 which has been secret shared by the honest party adversary has no information about the exactly value of q 4. From the view point of adversary it has only three shares of q 4, but those three shares could be three shares for any q 4 values from the field because q 4 has been secret shared through a polynomial of degree 3 which is randomly chosen/

But collectively parties know that out of these n number of q values n – t are not known to the adversary and remaining t they are shared by the adversary so anyhow it knows those t q values, but are those n – t values, q values which have been shared by the honest parties. So, P 4 does not know alone who else are the honest parties. It does not know whether P 1 is corrupt or P 2 is corrupt or P 3 is corrupt who else is corrupt.

We only have the knowledge that out of this n number of q values some n – t are random and not known to the adversary, but which n – t we do not know. So, now let us do the following let us do some magic let me define a function g which we will see is a linear function, which takes n values from the field and produces n – t values from the field. Namely it will take the input values q 1 to q n which have been secret shared by P 1, P 2, P 3, P 4 respectively.

And it will produce n – t number of output values. How exactly the output r 1 to r n – t are computed from the inputs. We imagine the polynomial fancy g which will have the degree n – 1 and passing through the points b 1 q 1, b 2 q 2, b n q n. Why its degree will be n – 1? Because we are given n number of points and we can always a fix a polynomial of degree n – 1 through those n points and these n points are unique because we have selected the beta components here distinctly.

(Refer Slide Time: 34:33)



Generating Shared Multiplication-Triples: Stage I

Rand-Extract

$\beta_1 \neq \cdots \neq \beta_n \neq \gamma_1 \neq \cdots \neq \gamma_{n-t} \neq 0$ $\quad t = 3$

- Each P_i (n, t) Shamir-shares a $q^{(i)} \in_r \mathbb{F}$
- $n - t$ values among $(q^{(1)}, \cdots, q^{(n)})$ are random from the view-point of adversary
  - $g : \mathbb{F}^n \rightarrow \mathbb{F}^{n-t}$ $\quad g(q^{(1)}, \cdots, q^{(n)}) \overset{\text{def}}{=} (r^{(1)}, \cdots, r^{(n-t)})$
  - $\mathcal{G}_{(q^{(1)},\cdots,q^{(n)})}(Z)$: unique degree-$(n-1)$ polynomial through $(\beta_1, q^{(1)}), \cdots, (\beta_n, q^{(n)})$
  - $r^{(i)} \overset{\text{def}}{=} \mathcal{G}_{(q^{(1)},\cdots,q^{(n)})}(\gamma_i)$ $\quad \nearrow f \quad q^{(1)}, q^{(2)} \ldots q^{(n)}$
  - Function g is a linear function

$q^{(1)} \in_r \mathbb{F}$ $\quad \mathcal{Q}^{(1)}(Z) \in_r \mathcal{P}^{q^{(1)},3}$	$q_1^{(1)}$	$q_2^{(1)}$	$q_3^{(1)}$	$q_4^{(1)}$
$q^{(2)} \in_r \mathbb{F}$ $\quad \mathcal{Q}^{(2)}(Z) \in_r \mathcal{P}^{q^{(2)},3}$	$q_1^{(2)}$	$q_2^{(2)}$	$q_3^{(2)}$	$q_4^{(2)}$
$q^{(3)} \in_r \mathbb{F}$ $\quad \mathcal{Q}^{(3)}(Z) \in_r \mathcal{P}^{q^{(3)},3}$	$q_1^{(3)}$	$q_2^{(3)}$	$q_3^{(3)}$	$q_4^{(3)}$
$q^{(4)} \in_r \mathbb{F}$ $\quad \mathcal{Q}^{(4)}(Z) \in_r \mathcal{P}^{q^{(4)},3}$	$q_1^{(4)}$	$q_2^{(4)}$	$q_3^{(4)}$	$q_4^{(4)}$
$r^{(1)} \in_r \mathbb{F}$ $\quad \mathcal{R}^{(1)}(Z) \in_r \mathcal{P}^{r^{(1)},3}$	$r_1^{(1)}$	$r_2^{(1)}$	$r_3^{(1)}$	$r_4^{(1)}$

So, pictorially what I am saying here is the following. You imagine that these are the values shared by the respective parties. Right now they are not available in clear, every party just have a share of these values, but you can always assume that there is a polynomial of degree n – 1 passing this n distinct points. Now my output values r output values are computed as follows. I think in my mind that I extend this polynomial this g polynomial at n – t new points or I evaluate this G polynomial at n – t new points new values.

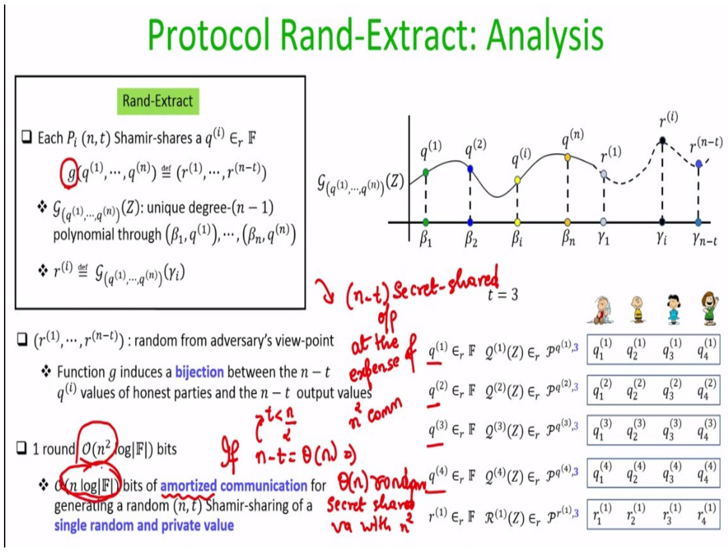
Why new values because now I am evaluating them at gamma values which are all different from the beta values. So, these are now distinct points, different points on the same G polynomial and that is the mapping g small g and my claim is that this function g is a linear function, it is a linear function of q 1, q 2 up to q n namely the output, the output r 1, r 2, r i r of n – t they actually are computed as per a linear function of q 1, q 2, q n.

Why linear function because again for this we have to go back to our earlier lecture where we discussed the properties of polynomial over field and there we discussed that we have already a polynomial of degree n – 1 passing through n distinct points then computing new points on the same polynomial is a linear function of the old points of the same polynomial. So, the points beta 1 q 1, beta 2 q 2, beta n, q n they are the old points on your G polynomial.

And the points gamma 1 r 1 gamma i r i gamma n – t r n – t they are the new points on your G polynomial. These new points are always a publically known linear function of the old points, but since the old points are secret shared namely the q components they are secret shared I can compute this new points also in the secret shared fashion and that is what we are going to do here.

So, in this specific demonstration since t = 3 and n – t = 1 I can imagine that there is a polynomial of degree 3 passing through this beta 1 q 1 beta 2 q 2, beta 3 q 3 and beta 4, q 4 and evaluate that polynomial at gamma 1 to obtain r 1. Now, since the function g is a linear function we can compute a secret sharing or r 1 by applying that linear function on secret sharing of q 1, q 2, q 3, q 4.

(Refer Slide Time: 37:54)



Protocol Rand-Extract: Analysis

Rand-Extract

- Each P_i (n, t) Shamir-shares a $q^{(i)} \in_r \mathbb{F}$

  $g(q^{(1)}, \cdots, q^{(n)}) \stackrel{\text{def}}{=} (r^{(1)}, \cdots, r^{(n-t)})$

  - $\mathcal{G}_{(q^{(1)}, \cdots, q^{(n)})}(Z)$: unique degree-$(n-1)$ polynomial through $(\beta_1, q^{(1)}), \cdots, (\beta_n, q^{(n)})$
  - $r^{(i)} \stackrel{\text{def}}{=} \mathcal{G}_{(q^{(1)}, \cdots, q^{(n)})}(\gamma_i)$

$\mathcal{G}_{(q^{(1)}, \cdots, q^{(n)})}(Z)$
$q^{(1)}$ $q^{(2)}$ $q^{(i)}$ $q^{(n)}$ $r^{(1)}$ $r^{(i)}$ $r^{(n-t)}$
β_1 β_2 β_i β_n γ_1 γ_i γ_{n-t}

- $(r^{(1)}, \cdots, r^{(n-t)})$: random from adversary's view-point
  - Function g induces a bijection between the $n - t$ $q^{(i)}$ values of honest parties and the $n - t$ output values
- 1 round $O(n^2 \log|\mathbb{F}|)$ bits
  - $O(n \log|\mathbb{F}|)$ bits of amortized communication for generating a random (n, t) Shamir-sharing of a single random and private value

$\searrow$ $(n-t)$ Secret-shared o/p at the expense of n^2 comm

If $\left(\because t < \frac{n}{2}\right)$ $n - t = \Theta(n) \Rightarrow$ $\Theta(n)$ random secret shared va with n^2

$t = 3$

$q^{(1)} \in_r \mathbb{F}$ $\mathcal{Q}^{(1)}(Z) \in_r \mathcal{P}^{q^{(1)},3}$	$q_1^{(1)}$	$q_2^{(1)}$	$q_3^{(1)}$	$q_4^{(1)}$
$q^{(2)} \in_r \mathbb{F}$ $\mathcal{Q}^{(2)}(Z) \in_r \mathcal{P}^{q^{(2)},3}$	$q_1^{(2)}$	$q_2^{(2)}$	$q_3^{(2)}$	$q_4^{(2)}$
$q^{(3)} \in_r \mathbb{F}$ $\mathcal{Q}^{(3)}(Z) \in_r \mathcal{P}^{q^{(3)},3}$	$q_1^{(3)}$	$q_2^{(3)}$	$q_3^{(3)}$	$q_4^{(3)}$
$q^{(4)} \in_r \mathbb{F}$ $\mathcal{Q}^{(4)}(Z) \in_r \mathcal{P}^{q^{(4)},3}$	$q_1^{(4)}$	$q_2^{(4)}$	$q_3^{(4)}$	$q_4^{(4)}$
$r^{(1)} \in_r \mathbb{F}$ $\mathcal{R}^{(1)}(Z) \in_r \mathcal{P}^{r^{(1)},3}$	$r_1^{(1)}$	$r_2^{(1)}$	$r_3^{(1)}$	$r_4^{(1)}$

So, that is a randomness extraction protocol here. So, now let us do the analysis of this randomness extraction protocol. So, first of all it produces n – t output values in a secret shared fashion and you see what is a communication, computation required we will analyze that later, but let us first try to analyze that why this new values r 1, r 2, r i, r of n – t they are random from the view point of the adversary.

Remember the output of the randomness extraction protocol that should be random from the view point of the adversary. Adversary should have just t shares for those output values nothing else and from those t shares it should not be able to infer anything. My claim here is that the function g that we have defined here it induces a bijection namely a one-to-one (()) (38:46) mapping between the n – t unknown q i values shared by the n – t honest parties.

And the n – t output values. What do I mean by that? So, again let us take this demonstration itself for understanding. We know that the value q 1, q 2, q 3 they are completely known to the adversary because adversary is controlling P 1 adversary is controlling P 2, adversary is controlling P 3. So, it knows completely the value q 1, q 2, q 3 secret shared by P 1, P 2, P 3, but what is the value q 4 from the adversaries point of view.

It could be any value since it could be any value hence it could be the any G polynomial which we are actually computing as part of the randomness extraction protocol that means adversary can fix or make an hypothesis about any candidate q 4 from the field for which it would have received three shares. Just based on three shares adversary cannot tell what exactly was the fourth shares and hence what was the fourth q value which the fourth party has secret shared.

That means for every candidate q 4 it could be any G polynomial and hence the G polynomial evaluated at gamma 1 could give any candidate element from the field. It could be either the polynomial G prime or it could be the polynomial G prime, prime or it could be the polynomial G triple prime and so on. For every candidate q 4 there is a corresponding G polynomial.

And that corresponding G polynomial evaluated at gamma 1 gives you the corresponding output and that is why since n – t input q values are not known to the adversary. As soon as adversary fix any candidate n – t q value in its mind that fixes the corresponding G polynomial in adversary's mind and hence it could be any n – t gamma values which the adversary's fixed polynomial would have produced in a secret shared fashion.

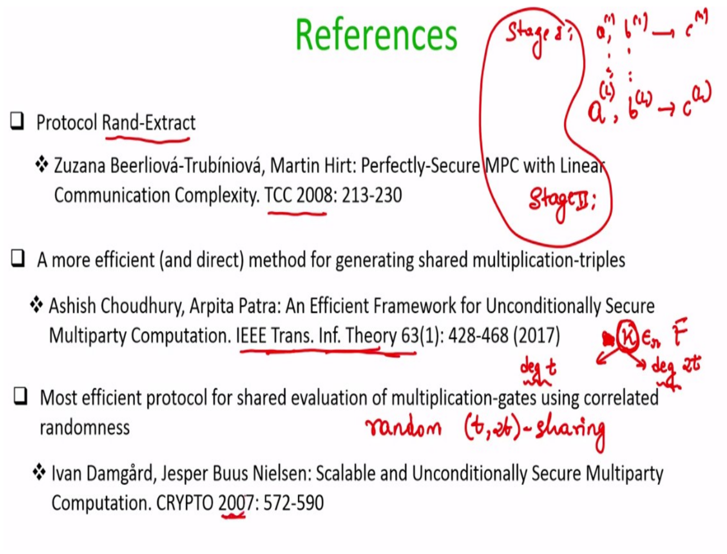
So, that ensures that indeed output values which are generated as per this randomness extraction process are indeed random from the viewpoint from the adversary, how many rounds are required here? Well, in this protocol each party just have to secret share a q value independently that will require one round and there are n instances of secret sharing each instance require a communication of n log F bits.

So, overall n square lot of bits are communicated, but by communicating n square log of bits we are able to generate how many secret shared outputs? We are able to generate n – t secret shared output. So, we can say that n – t secret shared outputs at the expense of n square communication of course in terms of field element. Now if n – t = theta n which will be the case for t less than n over 2.

Then basically we are saying then n – t will be theta n then we are basically generating theta n random secret shared values with n square communication that means in amortized sense generating one random secret shared value requires a communication of this much linear communication linear in the number of parties in terms of field elements and that is why this is a very powerful protocol.

It tells you how to generate one random secret shared value with no party knowing the value of that random value at the amortized expense of n log F bits of communication.

(Refer Slide Time: 43:19)



References

- Protocol Rand-Extract
  - Zuzana Beerliová-Trubíniová, Martin Hirt: Perfectly-Secure MPC with Linear Communication Complexity. TCC 2008: 213-230
- A more efficient (and direct) method for generating shared multiplication-triples
  - Ashish Choudhury, Arpita Patra: An Efficient Framework for Unconditionally Secure Multiparty Computation. IEEE Trans. Inf. Theory 63(1): 428-468 (2017)
- Most efficient protocol for shared evaluation of multiplication-gates using correlated randomness
  - Ivan Damgård, Jesper Buus Nielsen: Scalable and Unconditionally Secure Multiparty Computation. CRYPTO 2007: 572-590

Stage I:
$a^{(1)}, b^{(1)} \rightarrow c^{(1)}$
$a^{(l)}, b^{(l)} \rightarrow c^{(l)}$
Stage II:
$\boxed{K} \in_r F$
deg t
deg 2t
random (t, 2t)-sharing

So, these are the references used in today’s lecture the randomness extraction protocol has been taken from this beautiful work. There is of course a more efficient method for generating the shared random multiplication triples namely the Beaver triples. You can find the details in this paper or a very high level what we do in this work is the following. If you consider the pre-processing phase protocol that we have discussed.

What we do there we are following a two stage approach stage 1. We first generate the a components and b components in a secret shared fashion. We generate all the a components and the b components in a secret shared fashion and then in stage 2 we are basically computing the c components. In this paper we show how we can combine both these two stages in a single stage.

Namely we do not generate first the ab components and then compute the c components rather we ask every party to secret share some candidate random multiplication triple and then we perform a randomness extraction on the multiplication triples themselves. So, the details are available in this work and as I said earlier that there is no single method for generating correlated randomness.

There can be several ways of generating different form of correlated randomness. So, this paper in 2007 introduced a very nice form of correlated randomness which is called as random t, 2t sharing and what is a random t, 2t sharing. So, if I say that a value r is both t as well as t, 2t sharing that means there is some random value say K with uniformly at random from the field with no one knowing the value of K.

And K is secret shared both through a degree t polynomial as well as through a degree 2t polynomial of course both of this individual polynomials have to be random polynomials that means there will be a random t degree polynomial whose constant term will be K and parties will be having respective shares on that polynomial and there will be another polynomial a random 2t degree polynomial, but its constant term will be again the same K.

And parties will be having their respective shares lying on that random 2t degree polynomial that means now parties will have two shares for the same value K. One share corresponding to t sharing another share corresponding to 2t sharing. Now assuming that this kind of values have been generated in the pre-processing phase this paper gives you the efficient method for shared evaluation of multiplication gates.

So, again due to interest of time I am not going into the details, but you can see this paper. Thank you.