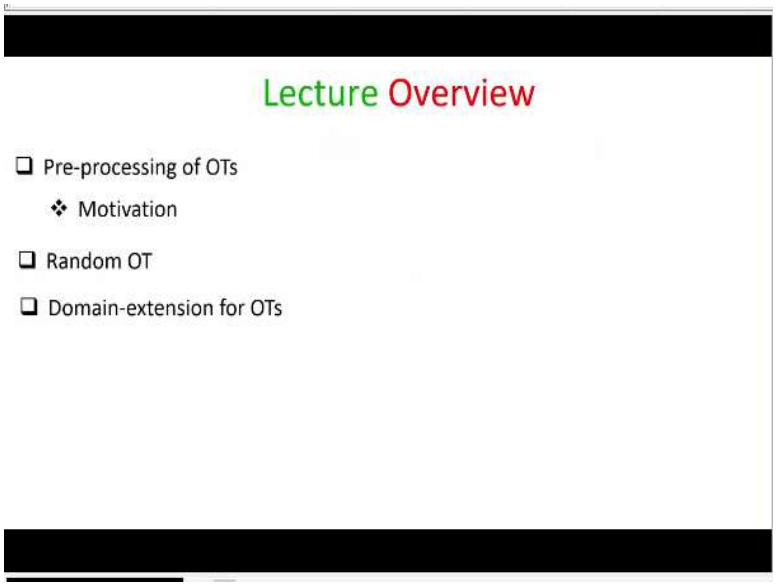
Secure Computation: Part I
Prof. Ashish Choudhury
Department of Computer Science Engineering
Indian Institute of Technology-Bengaluru

Lecture-45
Pre-Processing of OT

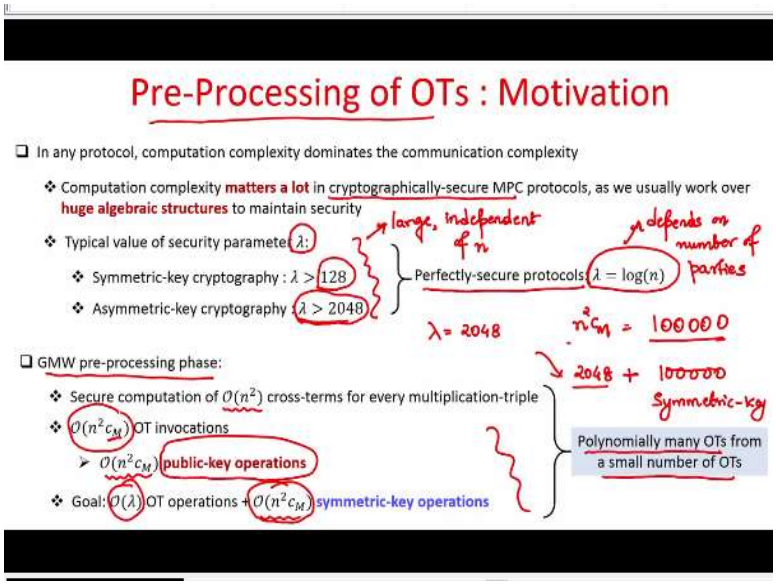
(Refer Slide Time: 00:31)



Hello everyone, welcome to this lecture. So, till now, we had seen GMW protocol and rule of oblivious transfer that is played in the GMW protocol specifically in the pre-processing phase. So, oblivious transfer in itself is a very important cryptographic primitive, it is used in lots of applications. So, we had already seen its application in the context of GMW, MPC protocol and later when we will discuss the Yao's protocol we will again see that oblivious transfer plays a very significant role.

So, what we will now do for next few lectures is the following, we will see in at how efficiently we can implement oblivious transfer protocols and this is done what by what we call as the pre-processing of OTs. So, the plan for today's lecture is as follows. We will first see the motivation for pre-processing of OTs and then we will see 2 simple ideas based on random oblivious transfer and domain extensions for oblivious transfer which will be later used for the pre-processing of OTs.

(Refer Slide Time: 01:39)



So, let us first see the motivation for pre-processing of OTs. So, the first thing to notice that in any protocol if you take any distributed protocol the computation complexity always dominates the communication complexity, because whatever the parties are supposed to communicate, they have to first compute it. And apart from whatever they are communicating, they also will be doing additional computations as part of the protocol.

So, computation complexity always dominates the communication complexity. And when it comes to cryptographically secure MPC protocols where we want to achieve security against polynomially bounded adversary then computation complexity matters a lot. This is because the primitives which are used inside the cryptographically secure MPC protocols. They work over huge algebraic structures to maintain security.

So, for instance if we take the oblivious transfer protocol we perform a lot of public key operations. And people who are familiar with public key cryptography specifically who have taken the foundations of cryptography course; they will know that the security of public key primitives are guaranteed, they are maintained only when certain computations are performed over huge algebraic structures.

So, called underlying hard problems, hard problem instances really become hard for any polynomially bounded adversary. So, in any cryptographically secure MPC protocol, we have this parameter λ which we call as the security parameter and depending upon what kind of cryptographic primitive the parties are using whether it is symmetric key primitive or whether it is asymmetric key primitive the value of security parameter varies.

And the security parameters may ensures how much security will be guaranteed by the underlying protocol and the value of the security parameter is dominated by the difficulty level of the so called underlying hard problems which define the security of the corresponding cryptographic primitives. So, if we are using symmetric key primitives say based on pseudorandom functions, pseudorandom permutations and so on or hash functions.

Then the value of the security parameter is 128. So, that means, if sender, receiver or the parties are using cryptographic primitives where they are using say a pseudorandom function then they have to use the key of size at least 128 bits. In the same way if they are using the parties or deploying hash function, then it has to be ensured that the hash function output has to be at least 128 bits.

So, that finding collision is difficult and so on, whereas if the parties are using asymmetric key primitives or public key primitives, like say Diffie-Hellman, based on Diffie-Hellman public problem, based on RSA problem and so on. And typically the value of a security parameter is 2048 bits, you can see that there is a significant gap in the value of the security parameter when it comes to symmetric key primitives and asymmetric key primitive.

And that is why we will prefer an MPC protocol, cryptographically secure MPC protocol where the parties are using as much symmetric key primitives as possible.**.** Of course whether indeed we can completely rule out asymmetric key primitive that is a natural question, but it turns out that we cannot do so, we have to deploy some form of asymmetric key cryptography, but idea will be to go for symmetric key primitives as much as possible.

On a different node if we consider the perfectly secure protocols which are secure against computationally unbounded adversary, there we do not use any cryptographic primitives. So, for instance, if we consider the BGW protocol, there the primitives which are used as secret sharing based on polynomial interpolation and so on. And there the value of security parameter is dominated by the field size.

And if we take semi secret sharing based MPC protocol and the field size has to be at least as large as the number of participants. So, there the security parameter λ is $\log(n)$; n is the number

of participants and now you can compare the value of the security parameter when it comes to cryptographically secure MPC protocols and information theoretically secure MPC protocols.

For the cryptographically secure MPC protocols the security parameter is large independent of the number of parties. And that means if we are deploying an MPC protocol for a small number of parties say 3 participants or 4 participants then even for such a specific setting where the number of parties is so small, if we are using a cryptographically secure MPC protocol we have to work or we have to work with a security parameter which is as large as 128 or as large as 2048.

Whereas, if we deploy a perfectly secure protocol then depending upon the value of number of parties, our security parameter can be very less or it can be very large. So, that depends upon the number of participants. So, now you might be tempting to say that since security parameter is very large for cryptographically secure MPC protocol, why we should go for cryptographically secure MPC protocols, why not just use perfectly secure MPC protocols?

So, recall that there is a trade off for perfectly secure MPC protocols you need to pay some price so for instance, we need to have perfectly secure channels among the participants and also we need a large portion of honest participants to achieve security. Whereas, if we go for cryptographically secure MPC protocols, then even if up to n – 1 participants are there we can still achieve security and we do not require perfectly secure channels pairwise secure channels among the participants.

So depending upon what you prefer, whether you want to work with a small security parameter, large security parameters or whether you want to tolerate more corruptions, less corruptions and so on. You have to decide what kind of MPC protocols you have to deploy. So, now, coming back to the question of pre-processing or OTs, if we see the GMW pre-processing phase then in the pre-processing phase, the parties have to do interaction while computing the secret sharing of multiplication triples.

And if we consider the n party case stand for each multiplication triple, there are order of n square number of cross dumps which has to be securely computed in a secret shared fashion. And for each cross term, we saw that 2 instances of oblivious transfer protocol are involved.

And as a result, if c_M is the number of multiplication gates in the circuit or a c_M is the number of multiplication triples which the parties want to generate.

Then we need n squared times c_M number of OT instances. And if you see the OT protocols, whether it is the bit OT or the string OT we deployed public key primitives, so for the bit OT, we constructed a bit OT based on the RSA function, we constructed the string OT based on the DDH assumption and in both the OT protocols public key operations are involved. Namely we have to compute or we have to perform modular exponentiation modulo a very large modulus.

At that size of the modules will be typically 2048 bits to ensure that indeed the discrete log problem, the DDH problem and its variants are hard or if we are using the RSA based OT protocol that we have to use prime numbers P and Q which are as large as 1024 bits. So, that the $\%\, N$ is 2048 bit modulus and then while computing the encryption, decryption we have to perform modular exponentiation modulo 2048 bit large compound modulus which is an enormous amount of computations.

So, that means, if we blindly implement a oblivious transfer protocols the way we have seen then the GMW pre-processing phase its cost will be proportional to performing $n^2 \cdot c_m$ number of public key operations and by public key operations I mean the modular exponentiation. Now, a natural question will be can I instead do the following? Can I do say order of λ number of oblivious transfer operations for the GMW pre-processing phase?

And using the output of those λ number of oblivious transfer operations whatever output receiver has and whatever input sender has based on those material we do the pre-processing for the GMW using only $n^2 \cdot c_m$ number of symmetric key operations or in general we want to ask the following question. Is it possible to get the effect of polynomially many number of OTs by performing only a small number of OTs plus some polynomial number of secret key operations or symmetric key operations?

And that is what we mean by pre-processing or OTs. Namely, we will ensure that the sender and the receiver who are going to be the 2 parties for the oblivious transfer they run some oblivious transfer protocols on some random inputs and then based on whatever data they generated through those OT instances after that they perform some symmetric key operations and then get the effect of polynomially many number of OTs.

If that is possible to do then that is a significant amount of saving in computation complexity. Because now, even though you wanted to have the effect of say $n^2 \cdot c_m$ number of oblivious transfers, you can get the effect of $n^2 \cdot c_m$ number of oblivious transfers by not performing a proportionate amount of public key operations. Rather you need to perform a proportional amount of symmetric key operations plus some constant number of oblivious transfer operations dominated by your security parameter.

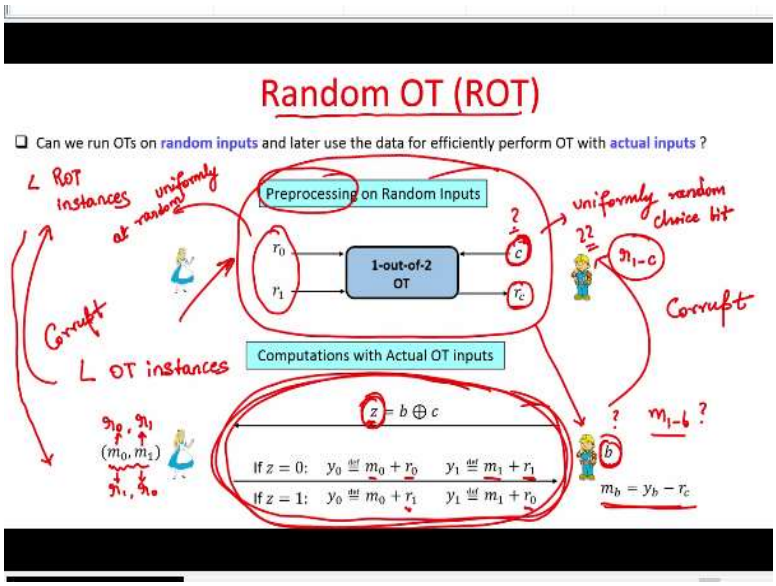
So, for instance, if I say take λ is equal to 2048 bits and say 1 is a very large quantity. Say 100,000. So, basically, if I want to generate 100,000 multiplication triples, then if I blindly run I have to perform order of 100,000 OT instances because there will be so many cross terms and if I deploy the oblivious transfer protocol 100,000 times then that is equivalent to performing 100,000 of the public key operations.

Through pre-processing of OTs what we want to do is the following. We want to do only say 2048 oblivious transfer operations, which will definitely require public key operations because for performing the oblivious transfer, we have to do the oblivious transfer plus 100,000 symmetric key operations. So, now, you will be saying that the number of operations remain the same.

In fact, now, we are doing more operations 100,000 symmetric key operations plus 2048 OTs, but the point is that the symmetric key operations are now performed using a small value of security parameter because since they are symmetric key primitives, their security will be guided or dominated by a different security parameter because they will be using pseudorandom function hash functions and so on those primitives for maintaining the security of those cryptographic primitives, we do not require a security parameter as large as 2048.

So, that is the saving here. And that is what we mean by pre-processing of OTs. And since OTs is going to play a significant role later again in the Yao's MPC protocol, if we can get this done namely if we can get the effect of polynomially many OTs by performing a small number of OTs plus some proportional amount of symmetric key operations then this will be a significant saving, because wherever you require a large amount of OTs to be executed involving the same sender and receiver then they can do the pre-processing of OTs and try to avoid the public key operations as much as possible.

(Refer Slide Time: 15:59)



That is a motivation. So, now we will see 2 quick ideas here which will be useful later for performing the pre-processing of OTs. So, the first idea here is what we call us random OT or ROT. And the question that we want to answer here is the following. Is it possible for the sender and the receiver to run OTs on random inputs? That means the sender's messages are going to be random, the receivers choice which is also going to be random.

And based on whatever receiver obtains as output in that random OT instance and whatever messages sender has used for those random OT instances, they can later use to add those data to actually perform the OT with their actual data for which they are supposed to do the OT but without running the oblivious transfer protocol. So, in some sense, we are trying to generate we are running we are asking the following question here.

We want to do some pre-processing on random inputs involving the sender and the receiver and whatever random data they generate, using that they can efficiently later implement actual oblivious transfer. So, that is what we mean by random oblivious transfer. So, here is how the random oblivious transfer protocol will work. So, we will first do the pre-processing here on the random inputs. So, by random inputs, I mean, the sender's messages or Alice messages, Alice is the sender, Bob is the receiver here.

So, Alice messages are r_0 and r_1 they are picked uniformly at random. So, if you are considering bit OT then r_0 and r_1 are random bits with equal probability they could be 0 or 1. Whereas, if we are considering string OTs say λ bit strings, where λ is a security parameter and r_0 and r_1

are random bit strings of length 1 bits and in the same way, this choice bits c is a uniformly random choice bit.

So, suppose sender and receiver are idle at the moment, they do not have the actual data on which they want to do the oblivious transfer. Say for instance, they are not yet started executing the GMW pre-processing phase; they are not generating the multiplication triples. So, what they can do is they can participate as many number of random OT instances as possible.

And these random OT instances will be executed using any of your oblivious transfer protocols that we had seen if it is a bit OT then it could be the RSA based OT, if it is a string OT then it could be the DDH based OT. So, let us take one such instance and by the property of the oblivious transfer Bob will receive the message r_c and we have the sender security maintained and receiver security maintained.

Now, suppose sender and receiver they have the actual data with respect to which they want to do the oblivious transfer. That means now Alice has the messages m_0 and m_1 ready one of which she wants to obliviously transfer to Bob depending upon the Bob's actual choice bit. Now I am not an m_1 they are not random messages. They are now application specific. So, for instance, if it is a GMW pre-processing phase protocol, then m_0 and m_1 will be say the shares of the multiplication triples available with Alice.

And Bob's choice bit will be share of one of those terms depending upon which cross Alice and Bob want to securely compute. So, they are no longer random inputs. But now we want to utilize the data which was involved in the pre-processing phase and do not want to run an actual oblivious transfer protocol involving Alice and Bob. So, let us see how it is possible to do this.

We want at somehow Bob should get a message m_b, Alice should not learn what message got obliviously and what message got transferred and we do not want to run a full-fledged oblivious transfer protocol to get this. So, here is how we can achieve the effect of the oblivious transfer. Bob will send a one time pad encryption of his actual choice bit b using the random choice bit with which he has participated in the random OT instance.

So, let us call that OTP encryption as z and if we consider a corrupt Alice will she learn anything about Bob's actual choice bit b? The answer is no. This is because during the random

OT instance from receiver security, Alice does not learn anything about c. And since c is acting as a one time pad, even though she is learning and OTP encryption of the Bob's actual choice bits in c is not known to Alice.

She does not learn anything about what is b. So, with almost probability half it could be 0 or 1. So, b is still hit. Now, what for the idea here is Alice now has to send both the messages m_0 and m_1 in some way. So, that Bob should be able to get back only the message m_b and nothing else, he should not get the other message. And the idea for doing that is Alice should use the messages r_0 and r_1, which she has used as the input for the random OT instance, to encrypt the messages m_0 and m_1.

Now the question is, can she directly use r_0 for encrypting m_0 and r_1 for encrypting m_1 or should she decide whether to use r_0 or r_1 for encrypting m_0 or m_1? And that depends upon the value of c, if the value of z is 0 and what does it mean when I say value of z is 0, the value of z = 0 means that whatever was a random choice bit of Bob in the random OT instance, is actual choice but b is the same.

That means if he was interested in the message, r_c, he is still interested in the message m_b, then only the value of z will be 0. That means when Bob participated in the random OT instance, his choice bit b was not yet ready. So he randomly participated with a choice bit c, it was 0 with probability half, it was 1 with probability half. Now his actual choice bit b is available with him.

So, it might be the case that he participated with c equal to 0 in the random OT instance. But now he wants to change his mind because he is actually interested in the message m_1, he was not aware what is going to be b, what exactly is input b going to be when he participated in the random OT instance. So, that is why the value of z could take 0 or 1 depending upon whether Bob is actually now changing his mind or not.

When he participated in random OT instance his choice bit b was not ready. Now his choice bit b is ready. And if he sees that his choice bit b is same as the choice bit c then z will be 0 otherwise equal to 1 and depending upon z equal to 0 or 1, Alice is going to use the pads r_0 and r_1 for encrypting m_0 and m_1 as follows. If z is equal to 0, then m_0 will be encrypted using r_0. So, r_0 will be acting as a pad for encrypting or masking m_0.

Whereas, r_1 will be used as a pad for masking m_1, whereas if z equal to 1, then r_1 will be used for masking m_0 and r_0 will be used for masking m_1. And now you can see that the order that the value of z decides whether it is r_0 followed by r_1 used for masking m_0 m_1 or whether it is r_1 followed by r_0 used for masking m_0 m_1. Because now if Bob's choice bit b is different from the choice bit c, he wants to ensure that the message m_b should be encrypted using r_c.

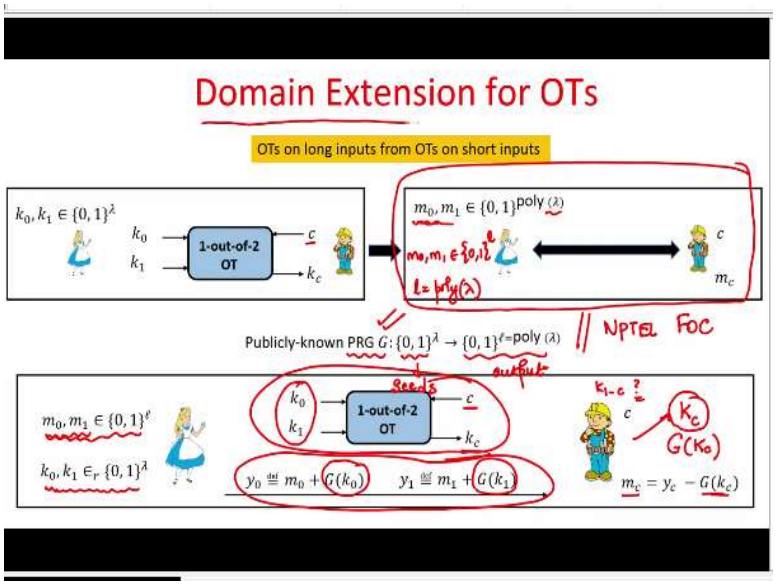
That is what we want to ensure. We want to ensure that somehow he has obtained the pad r_c and he wants to indicate to Alice that please encrypt the message m_b using r_c, he cannot directly tell that to Alice, because that will reveal the actual choice bit of Bob to the Alice, the way he can indicate to Alice in an obvious way is the following. He just sent an OTP encryption or a masking of b and c.

If b and c are same, then it is an indication that encrypt as per this order, whereas if z is equal to 1, then encrypted as per this order and Bob can now recover back the message that he is interested in by unmasking the pad r_c, which he got from the random OT instance from the ciphertext y_b. Now, let us see whether Bob learns the other message or not. So, for that, we have to consider the corrupt Bob.

And Bob is corrupt then he would not learn anything about the random input $r_1 - c$ that comes from Alice security or center security from the random OT instance. And as a result, he won't learn anything about the message $m_1 - b$, because the pad $r_1 - c$ is used for masking the message $m_1 - b$. So, now you can see for getting the effect of actual OT on the inputs m_0 m_1 and the choice bit b, we are not at all performing any public key operations. In fact, it is pure XOR based operation masking and unmasking operations that is all.

And that is why now this actual OT will be executed in a very fast fashion, because it does not require performing computing any modular exponentiation whatever public key operations need to be performed that we had already performed when the random OT instances are executed. So, this shows that if actually Alice and Bob are supposed to do L OT instances then they can do L ROT instances in parallel and later when the actual data for the L OT instances are ready use this mechanism L times.

(Refer Slide Time: 27:58)



Let us see another idea which is going to be used later for the pre-processing of OTs. This is called a domain extension for OTs. And here the idea is that can Alice and Bob perform or execute oblivious transfer on short involving short inputs and later use that output the data to perform oblivious transfer involving long inputs. More specifically, imagine that Alice and Bob they have performed, they have a mechanism to do oblivious transfer on λ bit inputs where sent Alice inputs or λ bit strings.

That means Alice input will be 2, λ bit strings k_0 k_1, Bob's choice which will be c and he will obtain the message k_c. Now, using this we want to do the following. We want to get the effect of oblivious transfer protocol where sender's message or Alice messages are now polynomial. They are now as long as polynomial factor of security parameters. So, you can imagine that m_0 and m_1 they are now say L bit strings, where L is some polynomial function of the security parameter.

But they do not want to run oblivious transfer involving now, L bit strings. They still want to run oblivious transfer involving λ bit strings, but somehow get the effect of oblivious transfer where Alice inputs are L bit strings. And this can be done very easily using a pseudorandom generator. So, assume we have a secure pseudorandom generator whose description is publicly known. And that pseudorandom generator takes λ bit seeds and it expand it is to an output of L bit.

It gives an output of length L bits. And its pseudorandom in the sense that if there is a polynomial time adversary who do not know the value of c, then it cannot figure out what is

output of the pseudorandom generator or formally in terms of indistinguishability random sample generated by this pseudorandom generator is indistinguishable from a uniformly random string of length L bits.

For any polynomial time distinguisher who is not given the value of the seeds with which the pseudorandom generator is invoked? So, if you want to know more about pseudorandom generators, see the NPTEL course on foundations of cryptography. So, bottleneck, bottleneck is if you do not know the value of the seed for the pseudorandom generator, you cannot tell what is the output of the pseudorandom generator.

That is roughly the security definition of pseudorandom generator; the output will be almost like a uniformly random L bit string, where each and every bit of the string can occur with probability almost half. So, assume that such a pseudorandom generator is available. And our goal is to get the effect of an oblivious transfer protocol where senders input are of size L bits, but we do not want to run a full-fledged oblivious transfer protocol, where senders inputs are L bit strings.

We want to run an oblivious transfer protocol where senders inputs are actually λ bit strings. So, how this can be done? So, sender's actual inputs are m_0 and m_1, one of which she wants to transfer to Bob depending upon his choice bit. To do that, what Alice does is the following. She prepares 2 random strings of length λ bits called them k_0 and k_1 and she expands the seed k 0 to generate a pad by using the pseudorandom generator and use it to mask the message m 0 and send the blinded version of m 0 to Bob.

And in the same way she used the other seed k_1 expands it and generate a pseudorandom string or pad of length L bits and using it blind the message m_1 and send a blinded version of m_1 to Bob. Now, Bob does not know the values of k_0 and k_1 and even though he knows the description of the pseudorandom generator, he cannot find out the value of g of k_0. He cannot find out the value of g of k_1 and hence, he cannot unmask the pads from y_0 and y_0 and recover back m_0 and m_1.

But Bob is interested to get the message m_c. To get the message m_c if we can ensure that Bob somehow gets the case of seed and we are done because Bob can get k_c then he can himself expand the seed k_c by running the pseudorandom generator himself. And running the

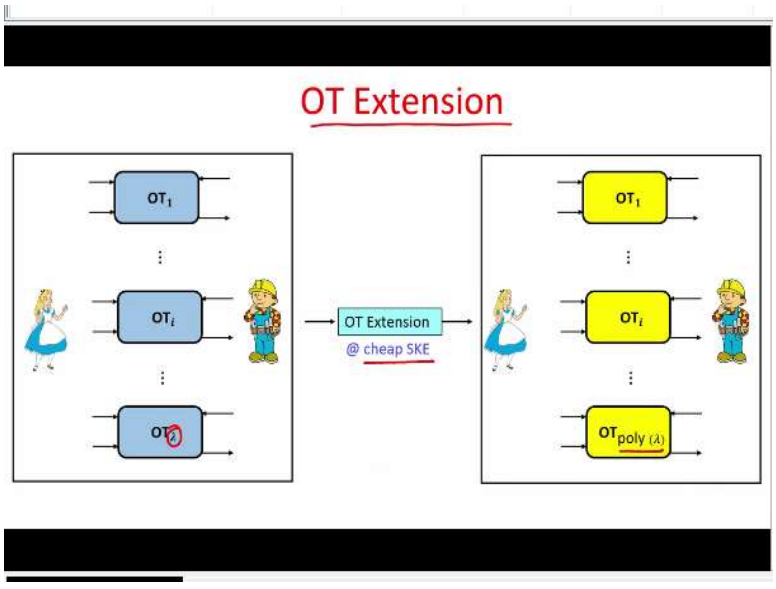
pseudorandom generator is an easy task because once the seed is known, the pseudo random generators output is a deterministic function and then he can recover back the message.

So, the question is how we enable Bob to get k_c? We cannot do the following that let Bob go and tell to Alice that please give me the key k_c because that will reveal that choice bit c to Alice. So, the way out here is the following. Let Alice participates in an OT instance where her inputs are the PRG seeds, k_0 and k_1. And Bob participates in the OT instance with the choice bit c and he gets only the seed k_c, which he is interested in. Now, once he gets the seed k_c, he himself can expand the seed k_c and unmask the pad from the ciphertext $y_c$and get back the message m_c.

Will a corrupt Alice learn anything about c? The answer is no. Will a corrupt Bob learn anything about the other message? Again, the answer is no because Bob does not learn the other PRG seed $k_1 - c$ and hence he cannot unmask the other ciphertext. Now what is the advantage of this whole process? The advantage of this whole process is that even though sender's inputs for of size L bits where L is significantly large compared to λ .

We do not require Alice and Bob to participate in an OT instance where Alice inputs are L bit strings. Rather, it is sufficient for Alice and Bob to participate in an OT instance where Alice inputs are only λ bitstrings and once that OT is executed the actual transfer of the message is performed by performing a very, very simple operation and this is how we can get that domain extension.

(Refer Slide Time: 35:20)



So, now, coming to the OT extension, what we want to get done through OT extension the thing that we want to get done through oblivious transfer OT extension is the following. Suppose there is a requirement where sender and receiver say Alice and Bob are supposed to get involved in several number of OT instances. Again, if you see the GMW pre-processing phase indeed it is the case where a pair of parties p_i, p_j, they are going to participate in n square c_M number of OT instances because they will be involved in securely computing those many number of cross terms.

So, through OT extension, we want to do the following, we will not want to run those many number of OT instances but rather we want the sender and the receiver to learn only λ number of OT instances. And then by performing some very, very simple and computationally efficient symmetric key operations, we somehow want to get the effect of these many number of OT instances.

Namely a polynomial factor of security parameter, those many number of OT instances, without actually running those many number of OT instances. And that is what is the whole goal of oblivious transfer extension And we will see in the next lecture that how we can get it done there. The domain extension is going to play a very crucial role and even we can use random OTs there as well. Thank you.