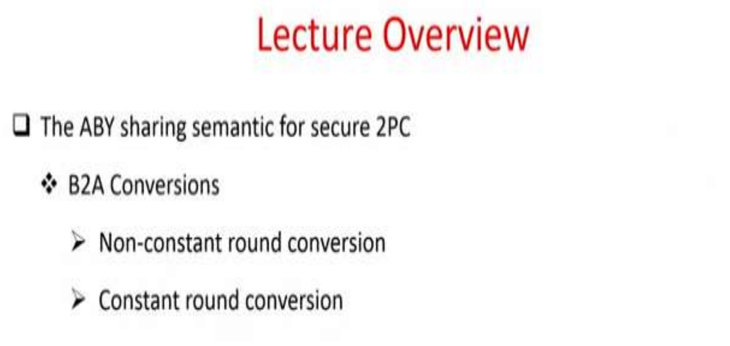
Secure Computation: Part I

Prof. Ashish Choudhury
International Institute of Information Technology, Bengaluru

Lecture - 57
The ABY Conversions Continued.

(Refer Slide Time: 00:33)

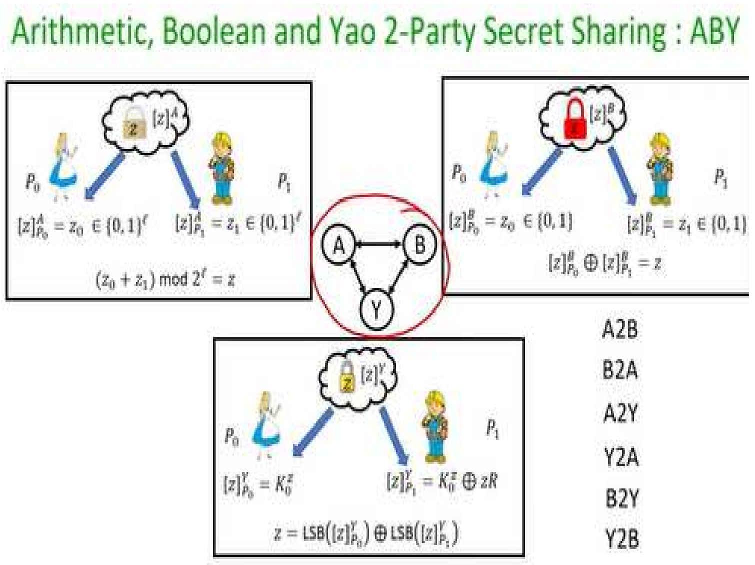


Lecture Overview

- ❑ The ABY sharing semantic for secure 2PC
  - ❖ B2A Conversions
    - ➢ Non-constant round conversion
    - ➢ Constant round conversion

Hello everyone, welcome to this lecture. So, we will continue our discussion regarding the ABY conversions, we had already seen few conversions the last couple of lectures. So, in this lecture we will consider the conversion from the Boolean sharing to the arithmetic sharing and we will see two conversions for this category, one which requires a constant number of routes and one which requires a non-constant number of rounds.

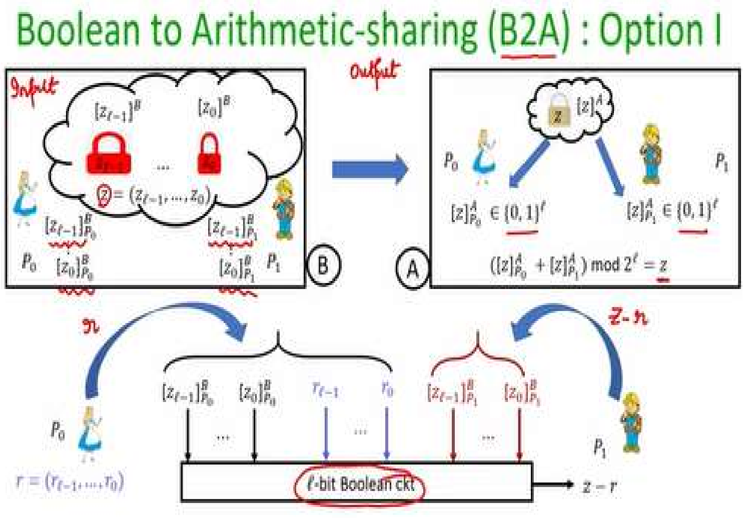
(Refer Slide Time: 00:56)



Arithmetic, Boolean and Yao 2-Party Secret Sharing : ABY
$[z]^A$
P_0
P_1
$[z]^A_{P_0} = z_0 \in \{0,1\}^\ell$
$[z]^A_{P_1} = z_1 \in \{0,1\}^\ell$
$(z_0 + z_1) \bmod 2^\ell = z$
$[z]^B$
$[z]^B_{P_0} = z_0 \in \{0,1\}$
$[z]^B_{P_1} = z_1 \in \{0,1\}$
$[z]^B_{P_0} \oplus [z]^B_{P_1} = z$
A
B
Y
$[z]^Y$
$[z]^Y_{P_0} = K_0^z$
$[z]^Y_{P_1} = K_0^z \oplus zR$
$z = \mathrm{LSB}([z]^Y_{P_0}) \oplus \mathrm{LSB}([z]^Y_{P_1})$
A2B
B2A
A2Y
Y2A
B2Y
Y2B

So, just to, quickly recap, we are currently looking into the conversions between three different secret sharing semantics for the two party case; arithmetic sharing, Boolean sharing and Yao sharing. We had already seen few of these conversions.

(Refer Slide Time: 01:14)



Boolean to Arithmetic-sharing (B2A) : Option I
Input
Output
$[z_{\ell-1}]^B$
$[z_0]^B$
$z = (z_{\ell-1}, \dots, z_0)$
$[z_{\ell-1}]^B_{P_0}$
$[z_0]^B_{P_0}$
$[z_{\ell-1}]^B_{P_1}$
$[z_0]^B_{P_1}$
P_0
P_1
B
A
$[z]^A$
$[z]^A_{P_0} \in \{0,1\}^\ell$
$[z]^A_{P_1} \in \{0,1\}^\ell$
$([z]^A_{P_0} + [z]^A_{P_1}) \bmod 2^\ell = z$
r
z-r
$r_{\ell-1}$
r_0
$r = (r_{\ell-1}, \dots, r_0)$
ℓ-bit Boolean ckt
$z - r$

Now, we want to see the Boolean to arithmetic conversion B2A. So, the input for this conversion is as follows, you have l bit number, which I call a Z whose binary representation is Z_0, Z_1, Z_{l-1} from LSB to MSB. Each of those bits are Boolean shared as per the GMW Secret sharing semantic. And, what is the output that we want from this conversion? We want the two parties to now possess arithmetic shares.

Namely to l bit numbers, such that if we add those two l bit numbers modular 2 power l output is equal to Z or the value is equal to Z. And in this whole process the value Z should remain private. That is what we want. So, let us see option number one to do this conversion, and the idea behind this option number one is as follows. We will ask one of these two parties say p 0 to pick a random l bit number l $_r$.

And our goal will be to ensure that Bob or the other party should obtain the value Z - r and Alice can keep our as her share for Z. So, Alice will keep on r as her share for Z, Bob will keep Z - r as his share of Z, and you can see if we add r and Z - r the output will be Z and in this whole process the value of Z should remain private. That means Alice even though she is picking r she should not learn the value of Z.

But somehow the value Z - r should be given to Bob through some protocol. So, how it is going to be insured? This is insured as follows. So, imagine you have an l bit Boolean circuit and his l bit Boolean circuit takes inputs both from Alice as well as from Bob. The inputs of Alice are her GMW shares for the bits of Z. So, she has her GMW shares for the bits of Z. So, they constitute one set of inputs from Alice.

And apart from that, it also takes the bits of the random number are picked by Alice also as the input. So, Alice has two sets of input, dual inputs here for this circuit and the input from Bob side for this circuit are Bob's GMW shares for the bits of Z. So, Bob has his GMW shares, so this l bit Boolean circuit takes those shares as inputs from Bob. And what is output of this Boolean circuit? The output of this Boolean circuit is the value of Z - r only for Bob.

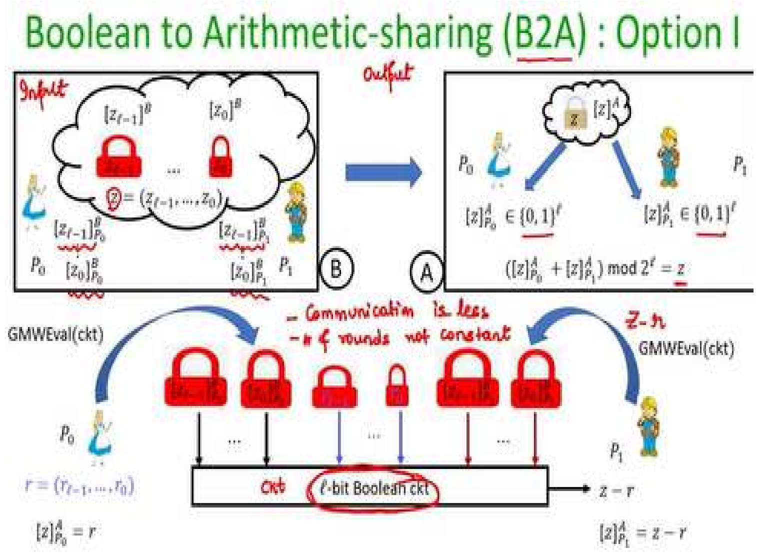
That means the output of this circuit is not going to be available both to Alice as well as Bob, the output will be only for Bob namely for the party who is not selecting the value r. Because If Z - r is also given as an output to Alice and then she since she has anyhow r, she will learn Z in the process. So, we require here a Boolean circuit, which when given these bits namely the Yao shares of Z from Alice and Bob.

And bits of r, a circuit should produce output Z - r for power. So, you can easily imagine what exactly that Boolean circuit will look like. The internal logic of this Boolean circuit will be as follows. It will first construct the bits of Z by exhorting the GMW shares of bits of Alice and Bob, which are fed as input to the circuit. And once it computes the value bits of Z, it will know the value Z.

And then, it has to basically add the bits of Z and the bits of r and produced output Z - r for Bob that is what is the internal logic for this Boolean circuit. Now, the idea behind this B to A conversion is to let the two parties Alice and Bob securely evaluate this circuit. And to securely evaluate this circuit what Alice is going to do is, she is going to share her parts of the inputs for this circuit by acting as a dealer.

So, she will be doing the GMW Secret sharing protocol, so she will act as a dealer and secret share each of our shares for the bits of Z and she will also act as dealer and secret share the bits of r. And what Bob is going to do is?

(Refer Slide Time: 06:50)



Boolean to Arithmetic-sharing (B2A) : Option I
Input
Output
$[z_{\ell-1}]^B$
$[z_0]^B$
$z = (z_{\ell-1}, \dots, z_0)$
$[z_{\ell-1}]^B_{P_0}$
$[z_{\ell-1}]^B_{P_1}$
P_0
$[z_0]^B_{P_0}$
$[z_0]^B_{P_1}$
P_1
B
A
$[z]^A$
$[z]^A_{P_0} \in \{0,1\}^\ell$
$[z]^A_{P_1} \in \{0,1\}^\ell$
$([z]^A_{P_0} + [z]^A_{P_1}) \bmod 2^\ell = z$
GMWEval(ckt)
- Communication is less
- # of rounds not constant
Z-r
$r = (r_{\ell-1}, \dots, r_0)$
ckt
ℓ-bit Boolean ckt
$z - r$
$[z]^A_{P_0} = r$
$[z]^A_{P_1} = z - r$

Bob is going to act as a dealer and he is going to secret share his inputs for this Boolean circuit. So, his inputs are his GMW shares for the bits of Z. So, he is going to act as the dealer and secret share those bits. And now, the inputs for this Boolean circuit are made available to these two parties in a secret shared fashion. What they are now going to do is, they are going to evaluate this circuit over the shares of the inputs as per the GMW circuit evaluation process.

So, I am calling this circuit as ckt. Now once their inputs so you can imagine that this is a Boolean circuit, which will have its OR gates or AND gates. What I am saying here is that, now Alice and Bob both will interact and evaluate each and every gate of this circuit ckt and reconstruct the output Z - r to what is Bob. If this is done then the arithmetic share for Alice will be r and the arithmetic share for Bob will be Z - r.

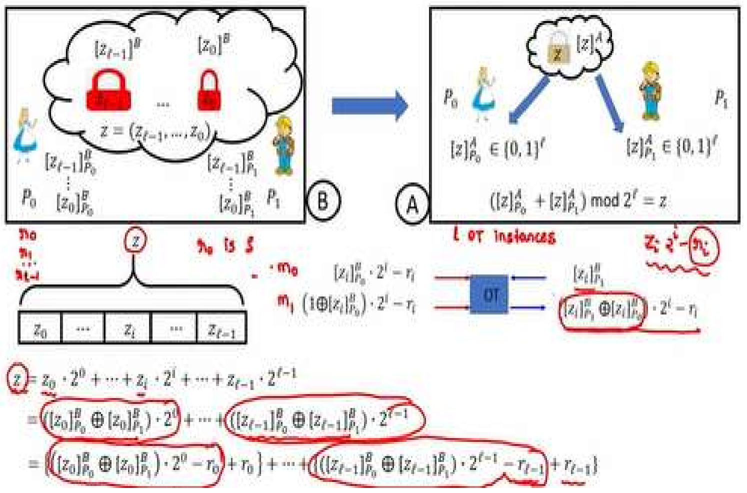
And in this whole process the privacy of Z is maintained, because neither Alice disclose anything about her GMW shares for the bits of Z, because they are secret shared, and in the same way Bob also does not reveal anything about his GMW shares of Z, because he has secret shared. So, that is option number one here. So, in this option, in this conversion from Boolean to arithmetic sharing, that communication overhead is less, communication is less, but number of rounds is not constant.

So, why the number of rounds is not constant? Because, if we go into the internal logic of this circuit ckt then, its multiplicative depth will be proportional to l and hence, by multiplicative depth I mean, there will be the number of layers of AND gate independent layers of AND gate and for each layer of AND gate, the parties need to interact and the number of layers of AND will be proportional to l.

So, that is why the number of rounds required here to do this conversion will not be a constant, it will depend on the value of l. So, if your l is large, then it will require a large amount of interaction among the parties. But during those interactions the communication will be less. So, if you are fine to allow the two parties interact, that means if the network latency is not very bad and you can go for this conversion. Because, the communication wise this conversion is very efficient.

(Refer Slide Time: 09:54)

Boolean to Arithmetic-sharing (B2A) : Option II



$[z_{\ell-1}]^B$
$[z_0]^B$
$z = (z_{\ell-1}, \dots, z_0)$
$[z_{\ell-1}]^B_{P_0}$
$[z_0]^B_{P_0}$
P_0
$[z_{\ell-1}]^B_{P_1}$
$[z_0]^B_{P_1}$
P_1
B
A
$[z]^A$
$[z]^A_{P_0} \in \{0,1\}^\ell$
$[z]^A_{P_1} \in \{0,1\}^\ell$
$([z]^A_{P_0} + [z]^A_{P_1}) \bmod 2^\ell = z$
ℓ OT instances
$z_i \cdot 2^i - r_i$
r_0
r_1
$r_{\ell-1}$
r_0 is \$
m_0 $[z_i]^B_{P_0} \cdot 2^i - r_i$
m_1 $(1 \oplus [z_i]^B_{P_0}) \cdot 2^i - r_i$
OT
$[z_i]^B_{P_1}$
$([z_i]^B_{P_1} \oplus [z_i]^B_{P_0}) \cdot 2^i - r_i$
z_0 | … | z_i | … | $z_{\ell-1}$
$z = z_0 \cdot 2^0 + \dots + z_i \cdot 2^i + \dots + z_{\ell-1} \cdot 2^{\ell-1}$
$= ([z_0]^B_{P_0} \oplus [z_0]^B_{P_1}) \cdot 2^0 + \dots + ([z_{\ell-1}]^B_{P_0} \oplus [z_{\ell-1}]^B_{P_1}) \cdot 2^{\ell-1}$
$= \{([z_0]^B_{P_0} \oplus [z_0]^B_{P_1}) \cdot 2^0 - r_0 + r_0\} + \dots + \{([z_{\ell-1}]^B_{P_0} \oplus [z_{\ell-1}]^B_{P_1}) \cdot 2^{\ell-1} - r_{\ell-1} + r_{\ell-1}\}$

Now, let us see the other option of doing the Boolean to arithmetic conversion, where we will now require a constant number of rounds. And now you can immediately think that, since we are aiming for constant number of rounds somewhere, we have to either bring in oblivious transfer or we have to somehow invoke Yao's, we have to evaluate some circuit in according to Yao's protocol, these are the only options.

So, this option number two for doing the Boolean to arithmetic conversion goes as follows. So, imagine the value Z which we want to be secret shared has its bits Z 0 from LSB to MSB, Z 0, Z 1, Z i, Z l - 1. Neither Alice nor Bob knows the exact values of these bits, because they have GMW shares of these bits. Now, since Z is an l bit number and the bits of Z or Z 0 to Z l - 1, I can write Z as per this formula.

So, I take Z 0 multiply with 2 powers 0, Z add and all the way to $Z^I{}_{i\,2}$ and all the way to $Z_{l - 1}$ power l - 1. Now, this Z_0 can be expressed as the XOR of the GMW shares of Z_0 available with Alice and Bob, and outside you have 2 power 0. In the same way if I consider that term Z_i, Z_i can be expressed as the XOR of the GMW shares of Z i available with Alice and Bob and outside you will have 2^i.

And in the same way, $Z_{l - 1}$ can be expressed as the XOR of the GMW shares of $Z_{l - 1}$ available with Alice and Bob and outside you have $2^{l - 1}$. And now, let us look closely into each term here.

So, if I consider this first term, this first term I can rewrite as I subtract r_0 from this first term and add r_0 to the first term. So, since I am subtracting r_0 and adding r_0 perfect of r is not there and basically, I am getting the first term only.

So, I am not changing the contribution of the first term here. So, here r_0 is some random l bit number. And the same thing I do for each and every time, if I considers the last term, I am pick I can rewrite this last term as if I take the last term, subtract a random l bit number and add the same l bit number. So, these r_0, r_1, r_{l-1} are random l bit numbers. The idea here is each of these terms can be rewritten as I mask each term with a random l bit number.

And unmask it with a random l bit number, the masked value. So, the effect of masking is not. Now, our goal is to ensure that at the end of this Boolean to arithmetic conversion for this l bit number Z. We should have two l bit numbers; one part held by Alice, another part held by Bob. So, that if sum those two parts we get the number Z that is what we have to do. Now, if we look closely here, what we can do is the following.

We can ask Alice to pick all this l bit random number, so, we can ask Alice to pick r_0, r_1, r_{l-1}, let Alice pick all this values. And now within each curly bracket if we somehow ensure that, the masking of the terms with the corresponding r values go to Bob somehow, that means if I take the first curly bracket here and somehow, we ensure that Bob gets $Z * 2^{0 - r}{}_0$. From the second curly bracket Bob gets $Z_1 * 2^{1 - r 1}$.

And like that from the last curly bracket, somehow, we ensure that Bob gets $Z_{l-1} * 2^{l-1-\text{of}\, r}{}_{l-1}$. If we ensure that Bob gets these terms, then we have actually identified arithmetic shares of Alice and Bob here. Alice can keep her arithmetic shares to be the summation of all this random r values. And within each curly bracket, whatever portion we are ensuring Bob receives.

If Bob sums all those pieces, then that constitutes Bob's share for the number Z. And now, you can see that indeed if we sum Alice share and Bob's share, we indeed end up getting the values. That is the idea behind this conversion. Now, everything boils down that, how within each curly bracket, this mask thing gets transferred to Bob and in the process nothing about Alice GMW share is not and in the same way nothing about Bob's GMW shares for the bits of Z is learned.

That is what we have to ensure. So, let us see how exactly the ith term, the masking of ith term that we are, desiring here gets transferred to Bob. So, you can imagine that Alice and Bob participate in l OT instances. I am going to show you only the ith OT instance because you can imagine that in the same way the parties Alice and Bob are actually executing l OT instances. I am just going to show you the inputs with which they participate in the ith OT instance.

So, in the ith OT instance, Bob participates as a receiver, which is choice bit being the GMW share for the bit Z_i. And our goal is to ensure that at the end of the OT instance Bob should get $Z_i * 2^{i} - r_i$, where r_i is a random value picked by Alice. So, in this process, nothing about Z_i is learned by Bob, because r_i is a random value and $Z_i * 2^i$ is masked to this r_i, so that preserves the privacy of the term Z_i.

Now to ensure that Bob indeed receives this output through the OT instance, Alice will participate as sender. The third message m_0 an m_1 being these values. So, m_0 will be her GMW share for the bit $Z_i * 2^{i} - r_i$ and m_1 will be the complement of GMW share for $Z_i * 2^{i} - r_i$. Where r_i is a random number l bit number picked by Alice. So, now you can check that indeed, if Alice participates with these values of m_0 and m_1 and Bob participates with this value of selection bit.

Then from the correctness of the OT, Bob will receive this output. Say for instance, Bob's GMW Share is for the bit Z_i is 0, if it is 0 then he should get the message m_0. So, if his GMW share for Z_i is 0, then the XOR of his GMW share and Alice GMW share will give actually Alice GMW here and outside you have $2^{i} - r_i$ and that is precisely is the message m_0. Whereas, if Bob's GMW share for Z_i is 1.

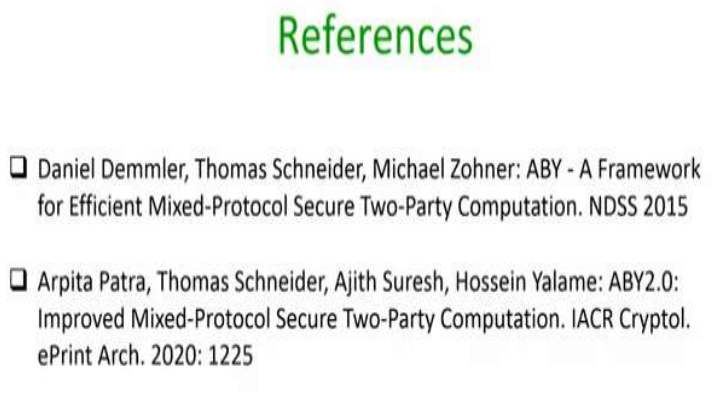
Then he should get the message m 1 and indeed if his GMW share is for the bit Z_i is 1, then the XOR of these two terms will be the complement of Alice's GMW share for the bit $Z_i * 2^{i} - r_i$. So, you can imagine that this OT instance is with respect to the ith term; since they are l terms here you can imagine that Alice and Bob participate in l independent OT instances. For each of the OT instances the random value r_i are not the corresponding random value r_i is picked uniformly at random by Alice.

And, that ensures that even though Bob receives to ith OT instance $Z_i * 2^{i-r_i}$. Since r_i are acting as a mask and they are picked independently of each other thing about the exact value of Z_i is learned in the process. And the privacy of receiver in this case ensures that through the in the OT instances nothing about the GMW shares of Bob learned by Alice. So, that is a constant round conversion.

Why constant round conversion? Because all these OT instances can be executed in parallel and OT is require a constant number of rounds. But OT is require us to perform public key operations, but we can take the help of OT extensions, namely all the B2A instances for which conversions are required during the computation, Alice and Bob can do the following. They will identify that at these points we have to do the B2A conversions.

So, they will identify the number of B2A conversions that are required say if it is L, capital L, then they can do the small l * capital L number of OT pre-processing and then later when the exact OTs are required. They can perform only symmetric key operations and do the corresponding B2A conversions. So, that way this requires a constant number of rounds of interaction.

(Refer Slide Time: 20:56)



References

- Daniel Demmler, Thomas Schneider, Michael Zohner: ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. NDSS 2015
- Arpita Patra, Thomas Schneider, Ajith Suresh, Hossein Yalame: ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. IACR Cryptol. ePrint Arch. 2020: 1225

So, these are the references used in today's lecture to discuss the B2A conversions. Thank you.