


**Secure Computation: Part II**  
**Prof. Ashish Choudhury**  
**Department of Computer Science and Engineering**  
**Indian Institute of Science, Bengaluru**

**Lecture - 55**  
**ICP from Information-Theoretic MAC: I**

(Refer Slide Time: 00:25)

## Lecture Overview



- ❑ Information-theoretic message authentication codes (MAC)
  - ❖ Linearity property
- ❑ ICP from information-theoretic MACs
  - ❖ The case of honest signer
  - ❖ Zero-knowledge consistency check between MAC tags and keys

Hello, everyone. Welcome to this lecture. So, in this lecture, we will first discuss about Information-theoretic message authentication codes and its Linearity property. And, using an information theoretic message authentication code, we will see how to construct an information checking protocol. So, we will first consider the simpler case of an honest Signer and then what are the challenges associated with that simpler protocol to deal with a potentially corrupt sign.

(Refer Slide Time: 00:55)

The slide is titled "Information-theoretic MAC" in green. It contains several mathematical expressions and handwritten notes in red. On the left, it says  $\mathbb{F} = \text{GF}(2^\kappa)$  with a red circle around  $2^\kappa$  and a note "Error prob of the primitive is at most  $2^{-\kappa}$ ". Below this, it says "Data:  $s \in \mathbb{F}$ " and "Tag  $\tau = \text{Mac}_k(s) = \alpha s + \beta$ ". On the right, it says "Mac key  $k = (\alpha, \beta) \in \mathbb{F}^2$ " and "description of a straight line over  $\mathbb{F}$ " with the equation  $y = \alpha x + \beta$ . There are small cartoon characters labeled P and V. A presenter is visible in the bottom right corner of the slide.

So, the way information theoretic message authentication code or MAC operates is as follows. So, we will have some finite field over which all the computations are performed, and we can work over the Galois field of size  $2^\kappa$  and this will guarantee that the error probability of the primitive is at most  $2^{-\kappa}$  how exactly that will be clear very soon. So,  $\kappa$  is one of your security parameters.

So, if you want to guarantee that the error in the primitive is at most  $\frac{1}{2^\kappa}$ , then you can choose a Galois field of size  $2^\kappa$  and perform all the operations in this message authentication code primitive over that finite field.

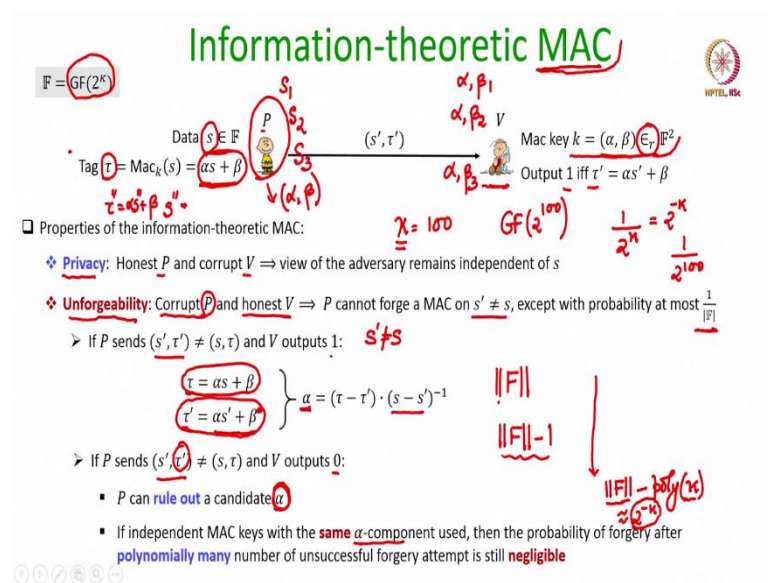
So, the way a MAC operates is as follows you will have two parties here say a Prover and a Verifier and Prover will have some data element, say an element from the field, and the Verifier will have a MAC key. And a MAC key here is a pair of random elements from the  $(\alpha, \beta) \in \mathbb{F}^2$  you can also interpret this MAC key as the description of a straight-line description of a straight line over  $\mathbb{F}$ , where  $\alpha$  represents the slope and  $\beta$  represents the intercept.

So, a straight-line equation will be of the form  $y = \alpha \cdot x + \beta$  where  $\alpha$  will be the slope and  $\beta$  will be the intercept. So,  $(\alpha, \beta)$  uniquely determines a straight line here. Now, this prover will have a MAC tag denoted by  $\tau$  which will be an authentication of this data  $s$

with respect to the key  $k$  and the tag is a point on the straight line namely the value obtained by substituting  $x = s$  in this straight-line equation, that is a MAC.

So, you can interpret the way this whole computation is performed here is as follows. The whole distribution of information is done is as follows. The straight-line equation is with the Verifier, but a data is not available with the Verifier, whereas the key is not available with the prover. And the key in this case is the description of the straight line, but prover has a point on that straight line. So, that is the way data is distributed here.

(Refer Slide Time: 04:25)



Now, where exactly this message authentication code will be useful? Later on, if prover wants to disclose or reveal its data  $s$  and also simultaneously get it verified by the Verifier; that means, there might be a possibility that the prover is corrupt, and he might want to reveal an incorrect data. The whole purpose of this message authentication code is to prevent a potentially corrupt prover from doing that and that too when the prover is computationally unbounded.

So, later, if the prover wants to reveal the data it can reveal the data and the corresponding MAC tag and now, the Verifier can accept the data revealed by the prover by itself recomputing the MAC tag with respect to the key and comparing it with the tag which the prover has provided. If the tag provided by the prover matches the tag recomputed by the Verifier, then the data is accepted namely the output is 1 otherwise the output is 0. So, that is the way typically an information theoretic MAC operates.

Now, let us see the properties achieved by this information theoretic MAC scheme. The first property is the privacy property and privacy is with respect to an honest prover and a corrupt Verifier. So, if the prover is honest, then until and unless the data is revealed to the Verifier. The Verifier does not learn anything about the data.

Of course, when the prover reveals, the data the Verifier will learn about the data. But, till the point the data is not revealed to the Verifier whatever information a corrupt Verifier has it reveals absolutely no information regarding prover's data.

This is because what data will be available with the Verifier it will only have the MAC key and the MAC key here is just a description of the straight line which is independent of the data  $s$  held by the prover. So, even if the Verifier is computationally unbounded it will not know what exactly the data  $s$  held by the prover is until and unless it is revealed to the Verifier. So, that is the privacy property achieved by this primitive.

Now, the important property which we want here is the unforgeability and the claim here is that if the prover is corrupt and the Verifier is honest, then the probability that a corrupt prover reveals an incorrect data  $s'$  and still the data gets accepted is at most 1 over the field size.

Now, assuming this property is true, what will be  $\frac{1}{|\mathbb{F}|}$ ?  $\frac{1}{|\mathbb{F}|}$  will be  $\frac{1}{2^\kappa}$  which is same as  $2^{-\kappa}$  which is a negligible quantity in your security parameter  $\kappa$ . So, say for instance, if you set  $k$  is equal to 100 and you have operating over a Galois field  $2^{100}$ , then the probability that a corrupt prover will be able to forge a tag on an incorrect data and still get it accepted is  $\frac{1}{2^{100}}$  which is a very small quantity.

So, of course, you can increase the value of  $\kappa$  if you want to make the error probability as small as possible. Now, why this claim is correct? So, imagine that the prover is corrupt. If the prover is honest, it will not try to reveal an incorrect data and the MAC tag. It will reveal the correct data namely  $s'$  will be  $s$  and  $\tau'$  also will be  $\tau$ .

But we are considering a scenario where the prover is corrupt. So, if the prover is corrupt, and suppose it sends an incorrect data and a MAC tag. Now, what is the probability that Verifier outputs 1 in this case. The Verifier outputs 1 in this case if the new tag namely  $\tau'$

indeed matches the indeed passes the verification done by the Verifier, that is  $\tau'$  is same as the tag for the data  $s'$  with respect to the key  $\beta$ .

So,  $\tau$  was the tag for the data  $s$  with respect to the key  $(\alpha, \beta)$ . So, prover had anyhow this tag on the data  $s$ . So, it knows that with respect to the unknown key  $(\alpha, \beta)$  this equation holds and now it is providing  $\tau'$  and it sees that the Verifier has accepted  $s'$ ; that means, it knows that for  $s'$  and  $\tau'$  with respect to the unknown key  $(\alpha, \beta)$  the second equation holds, right.

Both these conditions hold simultaneously; that means, the value of  $\alpha$  has to be this and the inverse of  $s - s'$  exists because we are considering the scenario where  $s'$  is not equal to  $s$ . That means, the probability that a corrupt party is able to forge MAC on an incorrect data is same as the probability with which it can find out the value of  $\alpha$ .

But  $\alpha$  is not known to the provers because the MAC key is known only to the Verifier, and we are considering the case when the Verifier is honest. So, forging a MAC tag on an incorrect data is same as guessing the value of  $\alpha$  and the probability that the prover can guess the value of  $\alpha$  correctly is  $\frac{1}{|\mathbb{F}|}$  because remember that both the  $\alpha$  as well as the  $\beta$  component of the MAC key are random elements from the field which are not known to the prover.

Also, a related property which will be useful later is the following if prover tries to send an incorrect data and a MAC tag and suppose, the Verifier rejects it; that means, the verification fails. Then it implies that the prover can rule out a candidate  $\alpha$ , why? Because prover would have tried to guess the value of  $\alpha$  in an attempt to compute  $\tau'$  and hoping that the  $\alpha$  which it has guessed is the right  $\alpha$  and the  $\tau'$  is the right tag on  $s'$ , but it turns out that suppose the guess was incorrect that was that means, the verification fails.

In that case the corrupt prover can rule out the guessed  $\alpha$  as a candidate  $\alpha$ ; that means, for him to begin with  $\alpha$  could have taken any value from the field; that means, there were field size number of candidates for the  $\alpha$ . It tried to forge a tag on some data and in that attempt, it guessed the value of  $\alpha$ , but then it finds that the Verifier has rejected that incorrect data and the corresponding tag; that means, it can say that whatever value of  $\alpha$  it has guessed that is not the right  $\alpha$ . That means, now the number of candidates for  $\alpha$  is reduced by 1 for this corrupt prover.

Now, why this property will be useful later looking ahead in our statistically secure MPC we will be using the information theoretic MAC where between the same prover and the Verifier the MAC keys will be used in the following way. The  $\alpha$  component of the MAC will be common for all the data held by the prover, but the  $\beta$  components will be independent.

And, in that scenario if the prover tries to guess the value of  $\alpha$  and try to forge MACs on incorrect data and every time his guess turns out to be incorrect it will be reducing the number of candidate values of  $\alpha$ . So, if we use this MAC and the way I have explained it here for polynomially many number of instances and there are polynomially many number of unsuccessful forgery attempt by a corrupt prover.

Then after polynomially many number of failed attempts, the number of candidate  $\alpha$  which prover will be left with will be  $|\mathbb{F}|$  minus whatever is the number of failed attempts. And this will be still an exponential quantity in your security parameter because if we take away polynomially many number of candidates from an exponentially large quantity.

Why exponentially large quantity? Because the field size is  $2^\kappa$  which is an exponential quantity in  $\kappa$ . So, if we take away polynomially many number of candidates asymptotically we are still left with exponentially many number of candidates.

Right now, you might be wondering that for the same prover and Verifier we cannot use the same MAC key across all the data held by the prover. Well, remember that prover will have one value on the straight line in the form of the MAC tag. So, it will not know the entire MAC key  $(\alpha, \beta)$ , but it will have one point on that straight line in the form of this MAC tag.

Now, imagine there is another data  $s''$  which is also authenticated with the same MAC key and say the resultant tag is  $\tau''$  and  $\tau''$  will be  $\alpha \cdot s'' + \beta$ . Now, the prover will be having two equations on this straight line through the tags  $\tau$  and  $\tau''$ , and hence it can easily find out both the  $\alpha$  component as well as the  $\beta$  component of the MAC key.

And, as a result of that a corrupt prover can now forge tag on any data of its choice, it will always pass the verification. So, that is why this entire MAC key cannot be reused for authenticating multiple data. What you can do is you can retain the  $\alpha$  component, but change the  $\beta$  component; that means, if  $s_1$  is one data authenticated with the key  $(\alpha_1, \beta_1)$ ,

you retain the same  $\alpha$  component, but change the  $\beta$  component and use it as a key to authenticate another data  $s_2$ .

If you have a third data you use the same  $\alpha$  component, but use an independent  $\beta$  component and you can use it to authenticate the data  $s_3$  for the prover and so on.

(Refer Slide Time: 16:49)

### Information-Theoretic MAC: Linearity

□ Consistent MAC keys: keys with the same  $\alpha$ -component

$$\left. \begin{array}{l} k_1 = (\alpha, \beta_1) \\ k_2 = (\alpha, \beta_2) \end{array} \right\} \text{consistent MAC keys}$$

□ Operations on consistent MAC keys: let  $k_1 = (\alpha, \beta_1)$  and  $k_2 = (\alpha, \beta_2)$

$$\left. \begin{array}{l} k_1 + k_2 \equiv (\alpha, \beta_1 + \beta_2) \\ ck_1 \equiv (\alpha, c\beta) \\ c + k_1 \equiv (\alpha, \beta + c) \end{array} \right\} c \text{ is a publicly-known constant from } \mathbb{F}$$


□ MAC will satisfy the **linearity property** if consistent MAC keys are used

$s_1, s_2$

$$\tau_1 = \alpha s_1 + \beta_1 \quad \tau_2 = \alpha s_2 + \beta_2$$

$$\tau_3 \equiv c_1 \tau_1 + c_2 \tau_2 = \alpha(c_1 s_1 + c_2 s_2) + (c_1 \beta_1 + c_2 \beta_2)$$

$$= \text{Mac}_{k_3}(c_1 s_1 + c_2 s_2)$$



$k_1 = (\alpha, \beta_1)$   
 $k_2 = (\alpha, \beta_2)$

$$k_3 \equiv c_1 k_1 + c_2 k_2 = (\alpha, c_1 \beta_1 + c_2 \beta_2)$$

MAC keys and tags for any linear function of data can be computed non-interactively

Now, we will be using a linearity property of information theoretic MAC in our statistically secure MPC. So, for that let us first define consistent MAC keys. So, imagine you are given two MAC keys  $k_1$  and  $k_2$  and if they have the same  $\alpha$  components then we say that these two keys are consistent MAC keys.

Now, let us define few operations on consistent MAC keys. So, if you have two consistent MAC keys and the sum of those two keys is defined to be a MAC key where the  $\alpha$  component is the same and the  $\beta$  component is the sum of the  $\beta$  components of the individual keys.

If  $c$  is a publicly known value from the field then  $c$  times the key  $k_1$  will be another MAC key with the same  $\alpha$  component and the  $\beta$  component being  $c$  times the  $\beta$  component of the key  $k_1$ . Whereas, the operation  $c + k_1$  will give you a MAC key where the  $\alpha$  component remains the same as in the key  $k_1$ , but the  $\beta$  component is the summation of the constant  $c$  with the  $\beta$  component of the key  $k_1$ .

So, if this is the way we perform operation on consistent MAC keys, then it turns out that the information theoretic MAC satisfies a very nice linearity property if consistent keys are used for a pair of prover and the Verifier. So, imagine prover has the data  $s_1, s_2$  and Verifier has a pair of consistent MAC keys and prover has the tags for  $s_1$  and  $s_2$  with respect to the keys  $k_1$  and  $k_2$  respectively.

Now, suppose prover computes this linear combination of the tags  $\tau_1$  and  $\tau_2$ . Then by expanding the value of the two tags and then rearranging the terms I get that the linear combination of the tags computed by the prover will be this value and suppose Verifier performs this operation on its MAC keys namely it computes the value  $c_1 \cdot k_1 + c_2 \cdot k_2$ .

Then as per the definition of operation on consistent MAC keys, the resultant MAC key will have the same  $\alpha$  component as the  $\alpha$  component of the keys  $k_1, k_2$ , but the  $\beta$  component will be  $c_1 \cdot \beta_1 + c_2 \cdot \beta_2$ . Mind it, no interaction has happened between prover and Verifier.

Now, what is the relationship between this linear combination of the tags  $\tau_3$  computed by the prover and the key  $k_3$  computed by the Verifier. It turns out that the tag  $\tau_3$  is nothing, but an authentication of the data  $c_1 \cdot s_1 + c_2 \cdot s_2$  with respect to the key  $k_3$ . Because, if indeed one would like to authenticate the data  $c_1 \cdot s_1 + c_2 \cdot s_2$  treating  $k_3$  as the key, then the corresponding tag will be the  $\alpha$  times data.

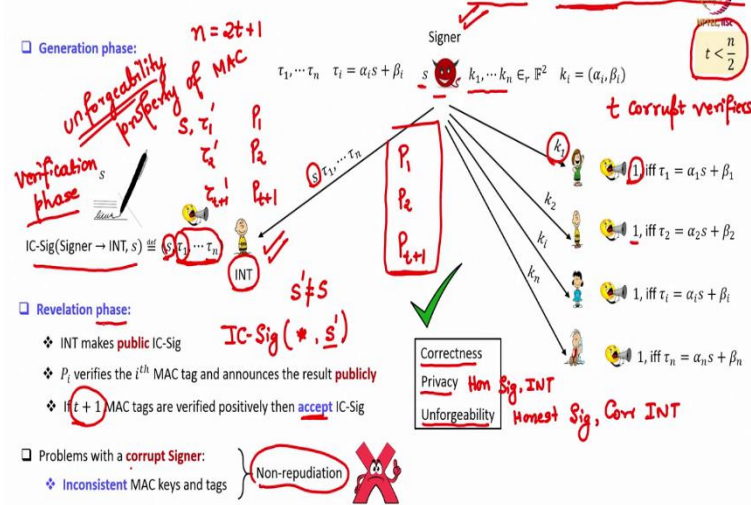
And data in this case is  $c_1 \cdot s_1 + c_2 \cdot s_2$  plus the  $\beta$  component of the key and the  $\beta$  component of the key is  $c_1 \cdot \beta_1 + c_2 \cdot \beta_2$  and this value is nothing but the linear combination of the tags computed by the prover. So, this shows that if we use consistent MAC keys between every pair of prover and Verifier, in some higher level protocol then the MAC keys and tags for any publicly known linear function can be computed without requiring any interaction between the prover and the Verifier.

And this is a very useful property looking ahead this will be useful in our statistically secure verifiable secret sharing scheme.



(Refer Slide Time: 21:30)

## ICP from Information-Theoretic MAC (Honest Signer)



So, now with information theoretic MAC let us see how we can proceed to design an information checking protocol and we will first start with a simple protocol simpler for the simpler case assuming that our Signer is honest. Remember that it need not be always the case and our final protocol will also take care of a potentially corrupt Signer, but to begin with we assume that we have an honest Signer.

And we are in the setting  $t < \frac{n}{2}$  again looking ahead this will be the optimal resilience for statistically secure MPC. So, we have a Signer with some data  $s$ , we have an intermediary and a set of Verifiers.

Now, the generation phase of this information checking protocol is as follows. The Signer it will pick  $n$  independent MAC keys and it will compute the tag for the data  $s$  with respect to each of these keys. To the intermediary it will give the data and the tags and that will be considered as the IC signature on the data  $s$  given to the intermediary. So, the data  $s$  along with the tags this entire vector is considered as the IC signature.

Now, the verification information given to the individual Verifiers is as follows. The first Verifier will get the MAC key  $k_1$ ; second Verifier will get the key  $k_2$  the  $i$ -th Verifier will get the key  $k_i$  and the  $n$ -th Verifier will get the key  $k_n$ . That is a generation phase. Now, since we are assuming an honest Signer this ICP will not require an intermediate phase to verify whether the Signer has distributed consistent data to the INT and the Verifier

because since the Signer is assumed to be honest, it will be distributing consistent data to the INT and the Verifiers.

Consistent data in the sense that the MAC tags which are given to the intermediary are indeed the correct tags with respect to the keys  $k_1, k_2, k_i, k_n$ . So, that is why it is safe to directly go to the revelation phase where if INT now want to if INT now wants to reveal the data  $s$  what it can do is the following it can make public the entire IC signature by using a reliable broadcast protocol and by making the IC signature public INT is basically making the value  $s$  public and all the MAC tags public.

Now, in response each Verifier will do the following. The Verifier  $V_1$  will only take the first tag from the signature which has been made public by the intermediary and it will check whether that is a right tag for the data  $s$  with respect to its MAC key  $k_1$ . If it is the case, then it says it broadcasts the value 1 in public otherwise it broadcast the value 0.

Now, it could be possible that even though intermediary has broadcasted the right IC signature, a corrupt Verifier unnecessarily broadcast the value 0 even though it is the even though the  $i$ -th tag is the right tag with respect to the key  $k_i$ . So, that is why to finally, decide whether the data produced by INT should be accepted or not, we check whether  $t + 1$  MAC tags are verified positively.

That means, whether at least  $t + 1$  parties have broadcasted the value 1 in response to the IC signature made public by the intermediary. If that is the case, then the overall output of the revelation phase is accept otherwise the output of the revolution phase is reject.

Now, it is easy to see that this simple ICP satisfies the correctness privacy and unforgeability properties. So, correctness property requires that if the Signer and INT are both honest then whatever signed data  $s$  intermediary reveals in the revelation phase it should get accepted and that holds because there will be at least  $t + 1$  honest parties honest Verifiers in the system who will always broadcast one in response to the IC signature made public by an honest intermediary.

The privacy property requires that if the Signer and INT are both honest, then till the revelation phase the adversary does not learn anything about the data  $s$ . So, since we are considering honest Signer and intermediary; that means, there are  $t$  corrupt Verifiers in

the system. Now, at the end of the generation phase what exactly will be the data available with that corrupt Verifiers?

Well, they will have only the MAC keys and those MAC keys will be independent of your data  $s$  which comes from the privacy property of the message authentication code. So, that shows that the privacy property is also achieved I stress that we require the privacy of the data only till the end of the generation phase because anyhow the regulation phase will make public the data  $s$ .

And, now let us see whether the unforgeability property is achieved. For this we have to consider an honest Signer and a corrupt intermediary, and we want to analyze here that what is the probability that a corrupt INT is able to forge an incorrect signature on a data  $s'$  which is different from  $s$ , but still, it gets accepted.

Now, there are  $t$  corrupt Verifiers in the system and if this corrupt INT tries to make public an IC signature on some data  $s'$  where  $s'$  is different from  $s$ ; that means, it has to also make public the corresponding tags with respect to data  $s'$ . Now, there are  $t$  corrupt Verifiers which can always say one in response to the tags which are made public by this corrupt INT.

But, as per the protocol we require at least one honest Verifier to say yes or to broadcast 1 in response to the tags which are made public by INT. So, for simplicity imagine that we are in the case where  $n = 2t + 1$  and say the first  $t + 1$  parties are honest.

We need at least of these  $t + 1$  parties to broadcast 1 in response to the data  $x$  prime which has been made public by INT. And, for doing that this corrupt INT has to forge the tag corresponding to the keys of at least one of these  $t + 1$  parties.

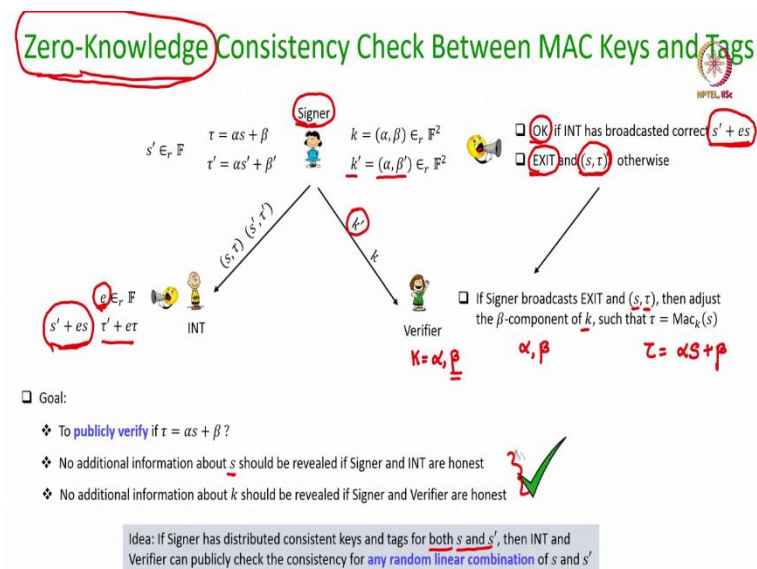
So, that means, either it has to make public a correct tag with respect to the first party's key or a correct tag with respect to the second party's key or it has to make a correct tag public with respect to the  $t + 1$ -th party. But the probability of doing that is very negligible very small because that comes from the unforgeability property. So, that means, if we assume an honest Signer, we can achieve the correctness privacy and unforgeability property.

But what if the Signer is corrupt? If the Signer is corrupt, then what it can do is it can distribute inconsistent MAC keys and tags to the to an honest INT and the honest Verifiers. And, because of that in the revelation phase if INT simply tries to produce the data  $s$  along with the tags received during the generation phase it will get rejected.

So, that is why, to deal with a potentially corrupt Signer we need an intermediate phase between the generation phase and the revelation phase to verify the consistency of the MAC keys and the tags. Because if we do not have that intermediate phase then the non-repudiation property which we require from ICP will not be achieved.

And, just to recap what is the non-repudiation property? The non-repudiation property means that at the end of this verification phase this intermediate phase which we are envisioning here, INT should have some signed data which is guaranteed to be accepted later in the revelation phase even if the Signer is corrupt. That property will not be achieved if we directly let the intermediary take the tags from the Signer and directly go to the revelation phase and try to reveal them.

(Refer Slide Time: 33:02)



So, for this verification phase we will require an intermediate zero knowledge consistency check between the MAC keys and the tags. And, let me explain the protocol with respect to one Verifier and later in the actual ICP we will see that this protocol is executed with respect to every individual Verifier.

So, what exactly we want to do here? So, we have the Signer who has distributed a data along with the tag to the intermediary and a Signer is claiming that tag is computed with respect to a key which is available with the Verifier. The intermediary cannot see the key, the Verifier cannot see the tag and now, the goal is to publicly check whether indeed intermediary got the tag with respect to the key held by the Verifier.

Of course, if the Signer is honest, it will distribute a consistent tag with respect to the key  $k$ , but if the Signer is corrupt it may deviate from the protocol and that is why we want to do this verification. We also want to do the verification in a zero-knowledge fashion in the and what does zero knowledge mean here? The zero-knowledge mean that during this verification process we are not willing to let any information about  $s$  to be revealed if Signer and INT are honest.

And, at the same time during this verification process, we do not want to reveal anything about the key if Signer and Verifier are honest. Of course, if INT and Verifier are both corrupt then the adversary knows the data, the tag, the key everything. So, you see both these requirements are with respect to different pairs of honest parties.

Now, the idea behind the verification the zero-knowledge consistency check is as follows. So, apart from the data  $s$  and a tag on the data  $s$  Signer will also select some random auxiliary data  $s'$  and a random auxiliary MAC key and it will compute a tag on this auxiliary data  $s'$  with respect to the key  $k'$ .

Now, this key  $k'$  is a consistent key; that means, its  $\alpha$  component is same as the  $\alpha$  component of the key  $k$  and the  $\beta$  component is picked independently. Now, apart from a data  $s$  and the tag on it, Signer now distributes additionally the data  $s'$  and tag on the data  $s'$  to intermediary and to the Verifier additionally the key  $k'$  is provided apart from the key  $k$ .

And, now, the idea behind is zero knowledge consistency check is the following that if Signer has indeed distributed consistent keys and tags for both  $s$  and  $s'$  stress for both  $s$  and  $s'$ , then any linear combination of  $s$  and  $s'$  and its corresponding tags can be verified in publicly with respect to any random linear combination where the linear combiners will be selected randomly and that will not be known to the Signer beforehand.

So, based on this idea, what the intermediary does here is the following. Once it receives the data  $s$  and  $s'$  and the tags on the data  $s$  and  $s'$  it picks a random linear combiner  $e$  which is a random element from the field and notice that Signer will have no information regarding  $e$  beforehand; beforehand means when it is distributing the data and the tag and the key to the INT and a Verifier. Only when Signer is distributed the data and the keys to the INT and Verifier respectively, this linear combiner is selected.

So, this linear combiner is made public and now, what it does it makes public a linear combination of  $s$  and  $s'$  and it is now a random linear combination you see because it is  $s' + e \cdot s$  where  $e$  is random. And a corresponding linear combination of tags on  $s$  and  $s'$  is also made public with respect to the same linear combination  $e$ . Now, before proceeding further we stress here that this random linear combination of  $s$  and  $s'$  does not reveal anything about  $s$ . We will touch upon this thing very soon.

Now, before Verifier responds anything what the Signer does here is the following. Signer performs some sanity check to verify whether indeed intermediary has behaved correctly or not. Behaved correctly in the sense, Signer will be able to see the random linear combiner  $e$ .

So, it knows that what random linear combination INT is supposed to now make public with respect to  $e$ ; that means, Signer will be able to check the value  $s' + e \cdot s$  and since anyhow it has distributed  $s$  and  $s'$  to INT, it should be able to verify whether INT has made public the right linear combination of  $s$  and  $s'$ .

If it sees that it has not made public the right linear combination of  $s$  and  $s'$ , then it simply instructs everyone to exit the consistency check by broadcasting an EXIT message and along with that it also makes public the data  $s$  and the corresponding tag which it would have given earlier to the intermediary.

So, there are two possibilities for Signer either it can say that ok, yes, I am fine with whatever INT has broadcasted; that means, it has made public the right linear combination of  $s$  and  $s'$  or it can say that no INT is not behaving correctly. So, I do not want to further continue our verification. I do not want to continue with the consistency check process and here is the  $s$  end tag that I would have given to INT earlier.

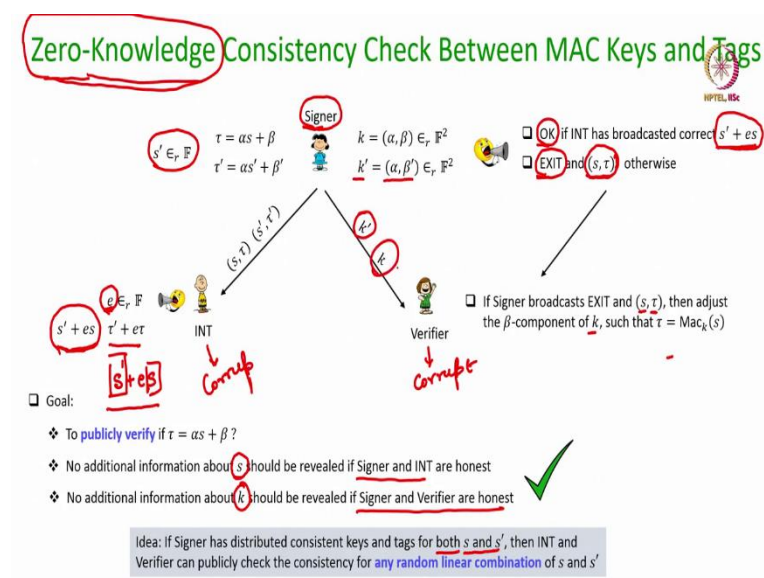
Now, if it is the second case; that means, if Signer has indicated to EXIT a protocol, then what the Verifier does is the following. It adjusts the  $\beta$  component of its key  $k$ . So, remember it has the key  $k$  which has an  $\alpha$  and  $\beta$  which it has received from the Signer. What it does is it adjusts the  $\beta$  component of the key, so that the tag which is now made public by the Signer it becomes the right tag it becomes the correct tag for the data  $s$  which has been now made public with respect to the adjusted key.

That means, earlier it has received  $\alpha$  and  $\beta$  and now, it now knows our value  $\tau$  which has been made public by the Signer. It keeps the  $\alpha$  component as it is and it changes the  $\beta$  component here in the box, so that the relation  $\tau$  is equal to  $\alpha$  times  $s$  plus the adjusted  $\beta$  holds.

So, now there are still some more steps left in this zero-knowledge consistency check, but before going into the details of those steps let us pause here and see whether we have achieved some of the required properties which we expect from this zero-knowledge consistency check.

If Signer and INT are both honest, then till now whatever values have been made public it does not reveal any information about  $s$ . What is made public about  $s$ , a random linear combination of  $s$  and  $s'$ ?

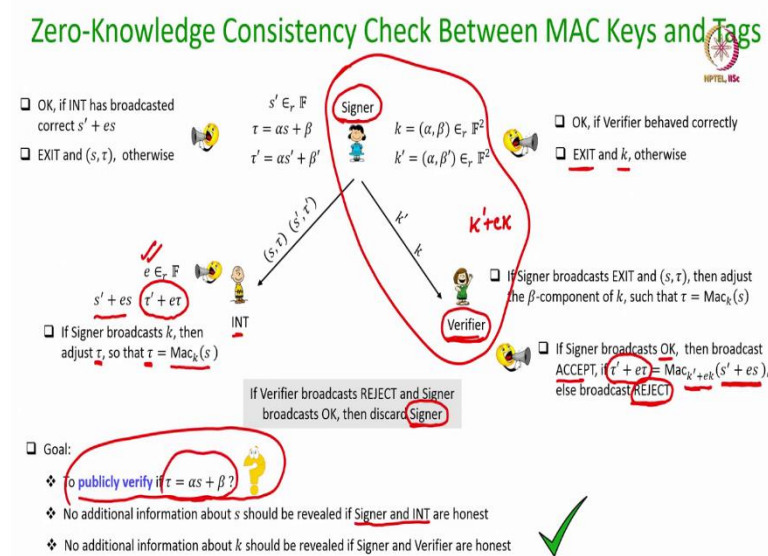
(Refer Slide Time: 42:24)



But,  $s'$  is also a random value and known only to INT; that means, if Verifier is corrupt it will be able to see the value  $s' + e \cdot s$ , but  $e$  anyhow will be also now learned by the Verifier because  $e$  is made public, but since  $s'$  is a random value this public value  $s' + e \cdot s$  does not reveal anything about  $s$ ; that means, the privacy of  $s$  is kept intact.

On the other hand, if the Signer and Verifier are honest, then no information about  $k$  is revealed to a corrupt INT because nothing about  $k$  has been made public till now.

(Refer Slide Time: 43:20)



Now, let us continue with the zero-knowledge consistency check. So, remember Signer could either make public an OK message in response to whatever linear combination INT has made public or it could have indicated to EXIT the consistency check.

If it has indicated to exist a EXIT a consistency check then the protocol would have halted here. But, suppose the Signer has broadcasted OK; that means, it says fine whatever linear combination INT has made public, it is a linear combination then it is a Verifiers turn to check whether the linear combination of tags which has been made public by INT is the right tag on this linear combination of the data  $s' + e \cdot s$  with respect to a MAC key which the Verifier can compute.

So, the Verifier has the keys  $k'$  and  $k$  and it anyhow knows the linear combiner  $e$  because that has been made public by INT. So, Verifier can perform the operations with respect to the consistent MAC keys, and it can compute the key  $k' + e \cdot k$  and then it can recompute



the tag on the data  $s' + e \cdot s$  with respect to this key and verify whether it matches the supposedly random linear combination of tags which INT has made public.

If that verification passes it says OK in public sorry it says ACCEPT in public, otherwise it says REJECT in public. And, now, in response to Verifier Signer has to say whether he is ok with Verifier's action or not. So, as Signer has said whether is ok with INTs action or not Signer has to now say whether it is ok with respect to Verifier's action or not. So, if it sees that Verifier has acted correctly, then it broadcast OK otherwise it indicates an EXIT from the protocol indicating that it feels that Verifier is not behaving correctly and hence it makes the key public.

If it makes the key public, then it is now INTs turn to adjust the tag that it has obtained earlier from the Signer, so that the adjusted or the recomputed tag becomes the right tag with respect to the secret  $s$  and the new key  $k$ . However, there is a case here that suppose the Verifier says REJECT; that means, it finds that the linear combination of the tags made public by INT is not the right tag for the linear combination of data with respect to the linear combination of keys.

But still signer says OK; that means, there is a conflict between what Verifier is saying and what Signer is saying. Verifier is saying no the system is inconsistent, but Signer is saying no, I am ok with Verifier's behaviour. If that is the case then that simply means that the Signer is corrupt because if the Signer is corrupt then if Verifier is corrupt if the Signer is honest and if the Verifier is corrupt and if the Verifier is unnecessarily accusing the system to be inconsistent, then Signer at the first place should have exited the protocol.

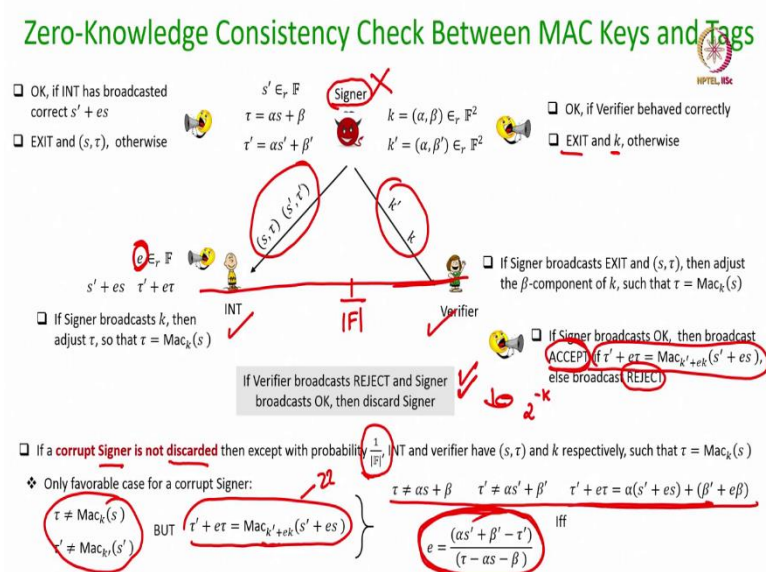
But, since Signer has not instructed to exit the protocol, but Verifier is saying that the system is inconsistent; that means, it is definitely the case that Signer is corrupt and that is why it is safe to discard the Signer and discard the Signer means that we now take some default data on the behalf of the Signer, default key with respect to the Verifier and default tag for the INT.

Now, let us see whether the required properties from the zero-knowledge consistency check are achieved. So, again if the Signer and INT are honest, in this full protocol no information is revealed about  $s$ . We had already argued about that with respect to whatever data INT makes public and now, Verifier does not make anything public regarding the data.

So, if Signer and INT are both honest the privacy of  $s$  is maintained whereas, if Signer and Verifier are both honest, then Signer will never make the key public; that means, it will never follow this EXIT path and hence no information about the key will be revealed.

Now, what is left is to argue that suppose at the end of this zero-knowledge consistency check, at no point Signer has indicated to exit the system and say the Signer is not discarded then does it guarantee that the Signer has distributed consistent tag and keys to INT and Verifier? So, that is the important property.

(Refer Slide Time: 48:58)



And now we are claiming here is that what we are claiming here is that if the Signer is corrupt and say it has distributed inconsistent data and tags and inconsistent keys to INT and verify; that means, we are considering the case where INT and Verifier are honest and Signer is corrupt, it has distributed inconsistent data.

But, suppose it is still not discarded, so, what I am claiming here is that if the Signer is corrupt and if it has distributed inconsistent data to INT and inconsistent keys to Verifier then except with probability  $\frac{1}{|\mathbb{F}|}$  it will be discarded. That means, with very high probability if the Signer is not discarded, then whatever data INT has, it is consistent with respect to the keys held by the Verifier.

So, suppose the Signer is distributed inconsistent data tag and keys, then the only favourable case for a corrupt Signer so that the Signer is not discarded is the following.

So, it has distributed inconsistent tags for  $s$  and  $s'$ , but suppose the random linear combiner  $e$  selected by INT is so is such that overall when Verifier checks the random linear combination of tags made public by INT, it matches the recomputed tag on the linear combination of data with respect to the linear combination of keys. That is quite possible.

If that happens then it will say accept and then everyone will believe that Signer has distributed consistent information to INT and Verifier. But, now what is the probability that even though the tag on  $s$  and  $s'$  are inconsistent with respect to keys  $k$  and  $k'$ , but the tag on  $s' + e \cdot s$  becomes consistent with respect to the key  $k' + e \cdot k$  that will happen if and only if all these three conditions hold simultaneously.

And this can happen if and only if the linear combiner is selected by INT in this whole system takes this value and you see  $e$  is well defined here because the denominator of  $e$  is not zero. And what is the probability that  $e$  takes this value? The probability that  $e$  takes this value is  $\frac{1}{|\mathbb{F}|}$ . Why? Because  $e$  is randomly selected from the field and Signer will have absolutely no idea about  $e$  till it is selected by INT and made public.

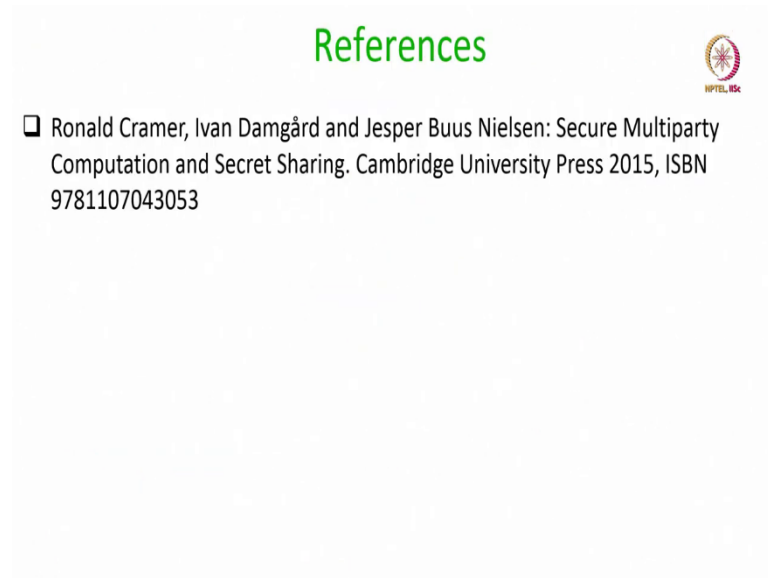
That means, when it is distributing the inconsistent data tag and keys to INT and Verifier Signer will not know beforehand what will be the challenge  $e$ , what will be the random linear combiner  $e$  that INT will pick. Only when it has distributed data key and tags, the INT is going to pick up the challenge combiner  $e$  and the probability that  $e$  takes up this favourable value is  $\frac{1}{|\mathbb{F}|}$ .

So, a corrupt Signer can just hope that even though it has distributed inconsistent data to INT and Verifier, it becomes lucky in the sense INT ends up picking this value of  $e$  because if INT ends up picking this value of  $e$  then even though it has Signer is distributed in consistent information, the honest Verifier will end up broadcasting except and no one will be able to catch this corrupt Signer.

But, what is the probability that  $e$  takes this value? It is  $\frac{1}{|\mathbb{F}|}$ ; that means, with probability  $2^{-\kappa}$   $e$  can take this value; that means, it is very unlikely that  $e$  will take this value. And hence Verifier will say REJECT. And, if Verifiers will say REJECT, then either Signer has to instruct to EXIT a loop in which case it will make the key public and if it makes the key public, INT will be adjusting its data and the tag.

And, hence as a result of that INT and Verifier will have consistent data or otherwise if Signer says OK, then this discard condition will be triggered, and Signer will be discarded from the system.

(Refer Slide Time: 54:04)



So, that ends the description of the ICP for the honest Signer case and the zero-knowledge consistency check. The reference for today's lecture is this book.

Thank you.