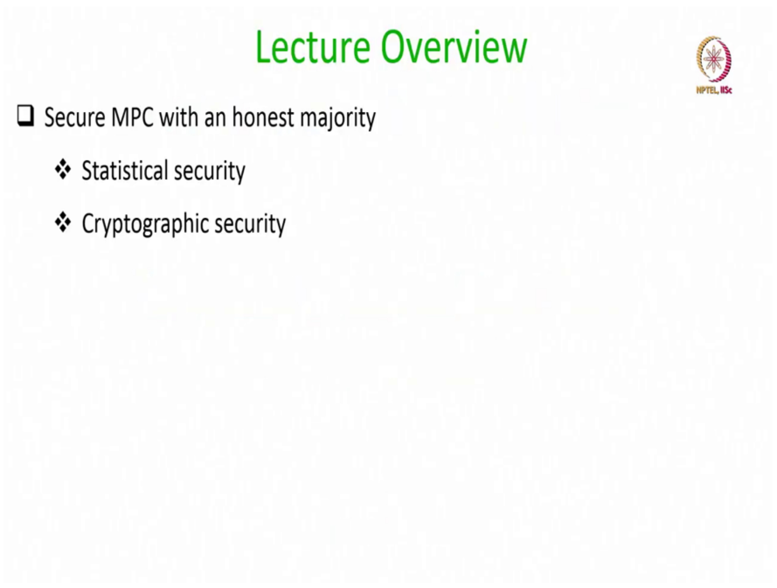
Secure Computation: Part II

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

Lecture - 54
Towards Secure MPC with an Honest Majority

Hello everyone, welcome to this lecture.

(Refer Slide Time: 00:25)

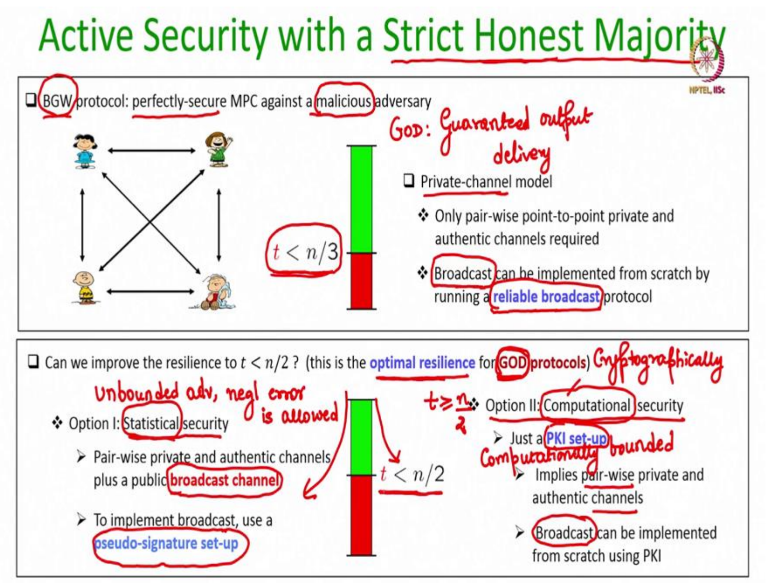


Lecture Overview

- Secure MPC with an honest majority
  - Statistical security
  - Cryptographic security

So, in this lecture we will discuss about Secure MPC in an Honest Majority setting, namely we will discuss about statistical security and cryptographic security.

(Refer Slide Time: 00:36)



Active Security with a Strict Honest Majority

- BGW protocol: perfectly-secure MPC against a malicious adversary

GOD: Guaranteed output delivery

$t < n/3$

- Private-channel model
  - Only pair-wise point-to-point private and authentic channels required
  - Broadcast can be implemented from scratch by running a reliable broadcast protocol

- Can we improve the resilience to $t < n/2$? (this is the optimal resilience for GOD protocols) Cryptographically
  - Option I: Statistical security
    Unbounded adv, negl error is allowed
    - Pair-wise private and authentic channels plus a public broadcast channel
    - To implement broadcast, use a pseudo-signature set-up

$t < n/2$

$t \geq \frac{n}{2}$

  - Option II: Computational security
    - Just a PKI set-up
      Computationally bounded
    - Implies pair-wise private and authentic channels
    - Broadcast can be implemented from scratch using PKI

So, we will focus on MPC with honest majority and by honest majority I mean majority of the parties are guaranteed to be honest. Our focus till now was on perfect security where adversary was computationally unbounded and we want all the properties to be achieved in an error free fashion. And we have seen the BGW protocol, of course, there are plenty of other protocols which are kind of variants of BGW protocol based on the principle of shared circuit evaluation which are perfectly secure.

And, all these protocols can tolerate up to n/3 corruptions, up to 33 percent corruption in the system. And, the setup that we require for these protocols is the private channel model; we do not require anything else. Private channel means we require every pair of parties should be able to communicate securely with every other party. Apart from that we do not require any other setup. Even though the protocols that we had seen they require broadcast, for simplicity we assume the presence of a broadcast channel, but that is just a simplifying abstraction, because we know that we can use reliable broadcast protocols like the phase king protocol and so on to emulate the effect of broadcast which are done as part of the BGW protocol. So, if you want perfect security, the maximum that you can tolerate, the maximum number of errors which you can tolerate is upper bounded by n/3.

A natural question is can we improve the resilience further, can we tolerate more number of faults? And if you want to achieve perfect security well that is not possible; that means, you have to degrade the security level of your protocols. So, what if I want to tolerate up to n/2 corruptions ok? Before, going into the answer for this question whether we can tolerate up to n/2 corruptions or not, this resilience of $t < n/2$ is the optimal resilience for guaranteed output delivery protocols.

So, this GOD here stands for Guaranteed Output Delivery. What does that mean? That means, even if up to t parties behave maliciously in the system they cannot prevent the honest parties from achieving the right function output; such kind of protocols are called as guaranteed output delivery protocols. So, it is easy to verify that the BGW protocol that we had seen, it is a GOD protocol.

Even, if up to t corrupt parties misbehave we have enough number of honest parties in the system who ensure that at the end of the protocol, the correct function outpost is learned by all the honest parties. It turns out that for any arbitrary MPC protocol with the GOD property the maximum number of faults which you can tolerate is $t < n/2$.

That means, if $t >= n/2$, then you cannot achieve GOD and intuitively this follows from the fact that if exactly 50 percent of the participants are corrupt, then you have half of the participants who are honest and half of the participants who are corrupt.

And, at the end of the protocol the corrupt parties might behave as if they are participating with some different set of inputs corresponding to which the function output could be something else compared to the 50 percent honest participates in the system. Its only when $t < n/2$; that means, the number of honest parties is at least one more than the number of corrupt parties, you can hope to achieve the GOD property.

So, now if you want to achieve the resilience of t less than n over 2 which is the maximum or which is the best you can do and achieve the GOD property, then there are two class of MPC protocols. The first class of MPC protocols is the statistically secure protocol where the adversary is still computationally unbounded, unbounded adversary, but negligible error in the protocol outcome is allowed ok. This is unlike your perfectly secure protocols where also the adversary is computationally unbounded, but we require absolutely no compromise on the security properties ok.

So, depending upon how critical is your application is, if it is a very very critical application where you are willing where you are not willing to compromise anything whatsoever in terms of security, you should go for perfect security, but then you have to pay a price. You can tolerate only up to 33 percent corruption but, if the security is not very critical, in the sense that it is fine if you have a non-zero error in the protocol outcomes; namely in terms of correctness, privacy.

But, that non-zero error is so small; it is so negligible that for most practical cases you can ignore it off. Then, you can go for statistically secure protocol where still adversary is allowed to be computationally unbounded. But, to design these protocols I need some additional setup on top of the private channel model. So, not only I require pairwise secure authentic channels, but I also require a broadcast channel.

And, now this broadcast channel in the setting of $t < n/2$ cannot be emulated by simply running a protocol over the point to point channels. So, recall during our discussion on reliable broadcast, we had discussed that if you want to get a statistically secure broadcast then you need some kind of setup apart from the pair wise secure channels. Namely, we require a pseudo signature setup.

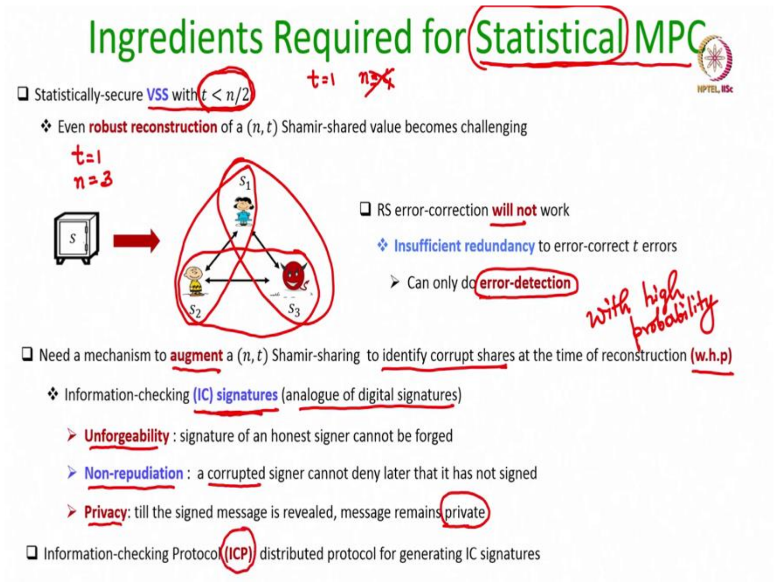
Option 2, will be to go for computationally secure or cryptographically secure MPC protocol, where adversary is now assumed to be computationally bounded, it is no longer computationally unbounded. So, here we have computationally bounded adversary, namely adversary is allowed to perform only polynomial amount of computation and here just a simple public key infrastructure setup is sufficient.

Namely, if we have a setup for encryption scheme and signatures using that we can not only implement a pair wise secure channels, we can also emulate the broadcast using the Dolev-Strong reliable broadcast protocols ok. So that means, the summary is that if you want to further improve the resilience of MPC protocols, but want to remain within the honest majority setting which of course, is a necessary condition for GOD protocols.

Then depending upon whether you have the pseudo signature set up or not, you can go either for statistical security and tolerate computationally unbounded adversaries or you can go for computational security and tolerate computationally bounded adversary. So, now, for the rest of our course we will see how to design statistically secure MPC protocols and how to design computationally secure MPC protocols.

The idea remains the same, we will be performing shared circuit evaluation for that we will require verifiable secret sharing scheme, methods for generating multiplication triplets and so on. But, since now a negligible error is allowed and I am no longer in the setting of t < n/3, the resultant sub protocols have to be changed.

(Refer Slide Time: 09:42)



Ingredients Required for Statistical MPC

- Statistically-secure **VSS** with $t < n/2$
  - Even **robust reconstruction** of a (n, t) Shamir-shared value becomes challenging
- RS error-correction **will not** work
  - **Insufficient redundancy** to error-correct t errors
    - Can only do **error-detection**
- Need a mechanism to **augment** a (n, t) Shamir-sharing to identify corrupt shares at the time of reconstruction **(w.h.p)**
  - Information-checking **(IC) signatures** (analogue of digital signatures)
    - **Unforgeability** : signature of an honest signer cannot be forged
    - **Non-repudiation** : a corrupted signer cannot deny later that it has not signed
    - **Privacy**: till the signed message is revealed, message remains private
- Information-checking Protocol **(ICP)**: distributed protocol for generating IC signatures

So, we start our discussion first with statistically secure MPC and for that we require first a statistically secure verifiable secret sharing scheme with t < n/2. And, there are some significant challenges to design a verifiable secret sharing scheme with t < n/2. Even, the basic task of reconstructing a secret shared value becomes challenging if you are in the setting of t < n/2; because now, the Reed-Solomon error correction will simply fail.

To demonstrate that imagine a scenario where t = 1 and n = 3 and say there is a value s which is secret shared through a 1 degree polynomial with three parties holding the shares s1, s2, s3 respectively and one of them is corrupt and your goal is to publicly reconstruct s. If there would have been 4 parties namely t = 1 and n = 4 parties, we know how to do it. We could have asked every party to exchange their share with every other party and then we can apply the Reed-Solomon error correction, where there will be sufficient redundancy to error correct one error.

But, now we no longer have 4 parties, we have only 3 parties out of which 1 could be corrupt and we do not know whether to take the shares s1, s2 and interpolate the 1 degree polynomial or whether to take the shares s2, s3 and interpolate the 1 degree polynomial or to take the shares s1, s3 and interpolate the 1 degree polynomial. That means, we do not have now sufficient redundancy as part of the Reed-Solomon error correction to error correct the corrupt share.

And, remember that no one will know the identity of the corrupt party here right. So, Reed-Solomon error correction will simply fail to give you back the correct 1 degree polynomial through which s is secret shared. You can only identify that some error has occurred; that means, you can only do error detection here ok. Error detection means you get the output that some error has occurred; that means, among these 3 shares definitely 1 share is corrupt.

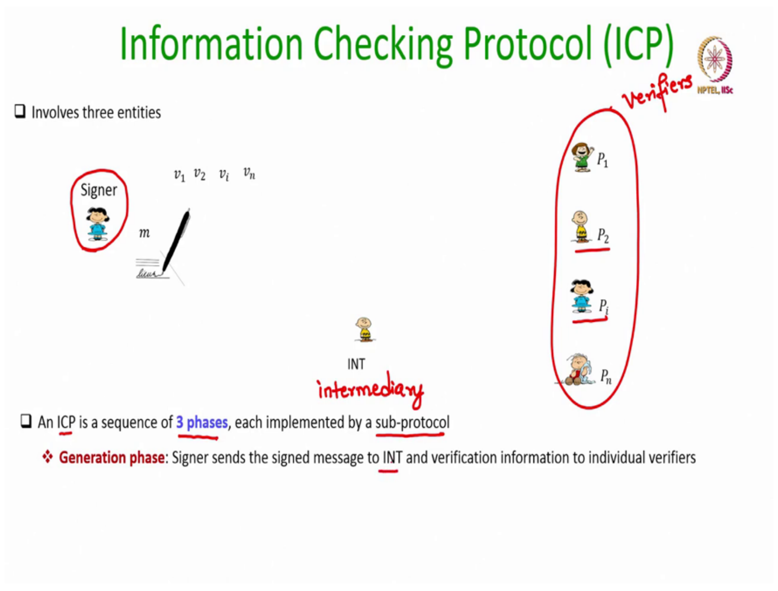
But, you will not learn which is that corrupt share whereas, error correction means you not only find out that error has occurred, you also identify where exactly the error has occurred. And, for doing the error correction we need more number of good shares, the right shares which is not happening in this particular example. So that means, we definitely need to augment our secret sharing, Shamir secret sharing to identify corrupt shares during the time of reconstruction where we are in the setting $t < n/2$.

But, fundamentally that is not possible to always identify the in to identify the corrupt shares, if you are in the setting of $t < n/2$. But, since we are in the statistical setting where a negligible error is allowed, we want a mechanism where we should be able to identify the corrupt shares with high probability which should be possible in the setting of $t < n/2$. And, that augmentation happens through information checking signatures, IC signatures; IC stands for Information Checking.

And, these signatures are analog of digital signatures except that all the properties are now achieved in the presence of a computationally unbounded adversary. So, like our digital signatures, we require the unforgeability property where the goal is that if the signer is honest then its signature cannot be forged on a message which it has never signed. We should have the non-repudiation property which should guarantee that, if the signer is corrupt and it has signed some message then later when the signature is publicly revealed, publicly shown, the signer cannot deny that it has not signed.

We need the privacy property as well from this primitive which should guarantee that until and unless a message signed by an honest signer is revealed, it should remain private. It should be available only with the entity who holds the signed message and this information checking signatures, IC signatures, they are generated using a distributed protocol which we call as Information Checking Protocol or ICP.

(Refer Slide Time: 14:46)



Information Checking Protocol (ICP)

- Involves three entities

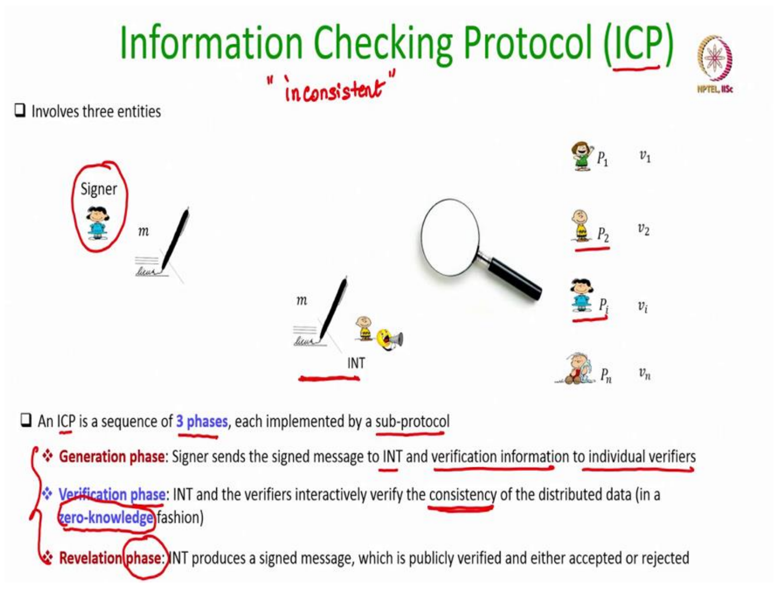
- An ICP is a sequence of 3 phases, each implemented by a sub-protocol
  - **Generation phase**: Signer sends the signed message to INT and verification information to individual verifiers

So, let us see the setting for Information Checking Protocol: ICP, who are the entities involved here and what are the sub protocols involved. So, here we have 3 classes of entities. We will have a signer who could be any of the n parties, who could be any designated party and it will have some message from some domain. And, there will be some intermediary INT.

Again, it could be any of the n parties and then the entire bunch of n parties will play the role of verifiers which will also include the signer as well as the INT, that is the way we are going to deploy the ICP protocol later ok. So, an ICP protocol or an ICP scheme, whatever you call it, will consist of 3 phases. Each of these phases is executed sequentially one after the other and each of these 3 phases is implemented by a sub protocol.

So, whenever I say information checking protocol, it basically has 3 sub protocols. The first phase is the generation phase, corresponding to which we will have a generation protocol. And, in this phase the signer will send the signed message to this intermediary INT.

(Refer Slide Time: 16:22)



Information Checking Protocol (ICP)

"inconsistent"

- Involves three entities

Signer
m
m
INT
P_1 v_1
P_2 v_2
P_i v_i
P_n v_n
NPTEL IISc

- An ICP is a sequence of **3 phases**, each implemented by a sub-protocol
  - **Generation phase**: Signer sends the signed message to INT and verification information to individual verifiers
  - **Verification phase**: INT and the verifiers interactively verify the consistency of the distributed data (in a **zero-knowledge** fashion)
  - **Revelation phase**: INT produces a signed message, which is publicly verified and either accepted or rejected

And, some verification information to individual verifiers. What exactly will be the signed message? How exactly the signed message look like? What exactly will be the verification information and so on? Well, that is determined by the exact instantiation of ICP. But, right now I am explaining you the abstract properties, abstract semantics and the required properties.

Now, once the generation phase is over, the next phase is the verification phase and the goal of the verification phase is to ensure whether a potentially corrupt signer has distributed consistent information to the intermediary and the verifiers. And, this is specifically to guarantee the non-repudiation property later. The way this ICP will be used is that later whenever INT would like to show the signed message, it will reveal the signed message publicly.

And, then the verifiers will verify individually the signed message with respect to their verification information. And, then based on certain conditions the signed message will be either accepted or rejected. Now, what a corrupt signer can do is, if the signer is maliciously corrupt and that is quite possible, because during the execution of the ICP protocol there could be up to t malicious corrupt parties which might either include the signer or the intermediary or the verifiers.

If the signer is maliciously corrupt, then it may distribute inconsistent signed message and verification information to the intermediary and honest verifiers respectively; and, that will violate the non-repudiation property; that means, INT will think that ok, it has got a proper signed message. And, then later when it attempts to reveal it to the verifiers, verifiers will say no, no you are corrupt, you are trying to forge signer’s signature, which is not the case, if the intermediary is honest.

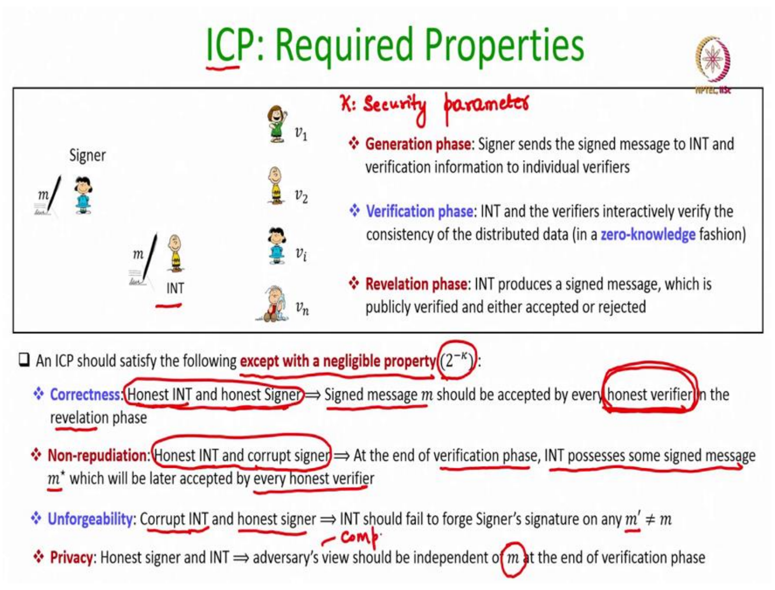
So, that is why after the generation phase, the intermediary and the verifiers along with the signer, they have to interact to guarantee that whatever information signer has given to intermediary, is consistent with the verification information available with the verifiers; consistent in the sense later it should get verified. One way of doing this consistency check is we ask the INT to make the message and the signature public and we ask the verifiers to make the verification information public.

But, that will simply violate the privacy requirement which we require from ICP. So, recall that one of the requirements of ICP is the privacy property which demands that until and unless INT does not show the signed message when it is required, the message should remain hidden. So, that is why this consistency check during the verification phase should happen in a zero knowledge fashion.

What does this zero knowledge mean here? The zero knowledge here mean that at the end of the verification phase INT and the verifier should come to the conclusion, that they hold consistent signature and verification information respectively. And, no additional information about the intermediaries message and the verification information should be revealed, if the signer and INT are honest, that is what the zero knowledge signifies here.

And, the third phase is the reveleation phase, where INT will now make public the signed message by broadcasting it. And, this message gets verified by the individual verifiers and depending upon certain criteria, it is either rejected or accepted. So, that is that is basically the syntax and semantics of the ICP.

(Refer Slide Time: 20:58)



ICP: Required Properties

Signer
m
INT
v_1
v_2
v_i
v_n

κ: Security parameter

- **Generation phase**: Signer sends the signed message to INT and verification information to individual verifiers
- **Verification phase**: INT and the verifiers interactively verify the consistency of the distributed data (in a **zero-knowledge** fashion)
- **Revelation phase**: INT produces a signed message, which is publicly verified and either accepted or rejected

- An ICP should satisfy the following **except with a negligible property** ($2^{-\kappa}$):
  - **Correctness**: Honest INT and honest Signer $\Rightarrow$ Signed message m should be accepted by every honest verifier in the revelation phase
  - **Non-repudiation**: Honest INT and corrupt signer $\Rightarrow$ At the end of verification phase, INT possesses some signed message m^* which will be later accepted by every honest verifier
  - **Unforgeability**: Corrupt INT and honest signer $\Rightarrow$ INT should fail to forge Signer's signature on any $m' \neq m$

    comp
  - **Privacy**: Honest signer and INT $\Rightarrow$ adversary's view should be independent of m at the end of verification phase

Now, let us see the required properties, what are the properties which we require from an ICP ok. So, we require the following properties to be achieved with very high probability; that means, all these properties should be achieved except with some negligible probability, that negligible probability is $2^{-\kappa}$ where κ is the security parameter. So, we can set the security parameter depending upon how much error we want to tolerate.

And, that will determine the size of the field over which the computations have to be performed while deploying or instantiating this ICP. So, the first property we require is the correctness property which should guarantee that if you have an honest intermediary and honest signer, then when the signed message is revealed during the revelation phase; it should be accepted by every honest verifier. That means, it should not happen that signer is honest, intermediary is honest and then when intermediary shows the message during the revealation phase; it is not accepted by the honest verifiers.

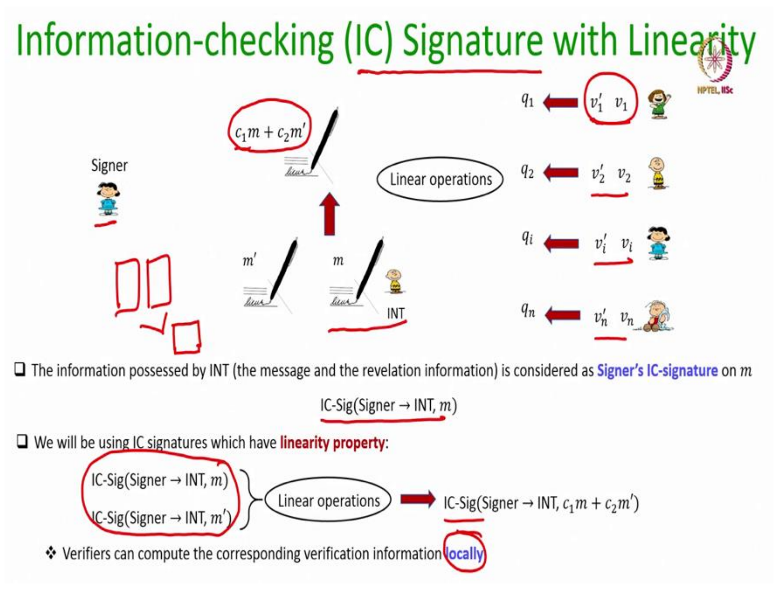
Why honest verifiers? Because, corrupt verifiers can always reject a correctly signed message; they may say oh I do not accept your message, this is fraud; fine they can always do that. But, the honest verifiers in the system, they should not reject a correctly signed message, that is the correctness property. The next property is the non-repudiation property which should hold for an honest INT and a corrupt signer. You see these properties are with respect to certain combinations of honest entities and corrupt entities.

So, the corrupt correctness property was for an honest INT and honest signer along with honest verifier. Non-repudiation means we are considering the case when signer now, wants to fool an intermediary by giving him some bogus signed message. So, that later when intermediary tried to reveal it during the revealation phase, it is not accepted. Non-repudiation says that should not happen; that means, once the verification phase is over and the zero knowledge verification has happened, at the end of the verification phase INT should possess a valid signed message. It need not be on m, it could be on $m^\star$ ok.

It could be such that signed m star when INT shows later during the revolution phase, it should get accepted by every honest verifier. We require the unforgeability property which should hold with respect to a corrupt INT and an honest signer. So, we require that if the signer is honest and if he has not signed m' for INT, then later if the INT is corrupt, it is malicious and try to reveal the signer's signature on m', where m' is different from m, then it should fail. All the honest verifiers should reject it.

And, the fourth property is the privacy property which demands that if the signer is honest and INT is also honest, then throughout the ICP protocol; that means, till the end of verification phase, whatever information adversary has learnt from t corrupt parties that should not help the adversary to learn anything about m. That means, the view of the adversary should be independent of m and this should hold even if the adversary is computationally unbounded. In fact, all the properties hold against computationally unbounded adversaries. So, these are the four requirements from an ICP.

(Refer Slide Time: 25:10)



Information-checking (IC) Signature with Linearity

- The information possessed by INT (the message and the revelation information) is considered as **Signer's IC-signature** on m

  IC-Sig(Signer → INT, m)

- We will be using IC signatures which have **linearity property**:

  IC-Sig(Signer → INT, m), IC-Sig(Signer → INT, m') } Linear operations → IC-Sig(Signer → INT, $c_1 m + c_2 m'$)

  - Verifiers can compute the corresponding verification information **locally**

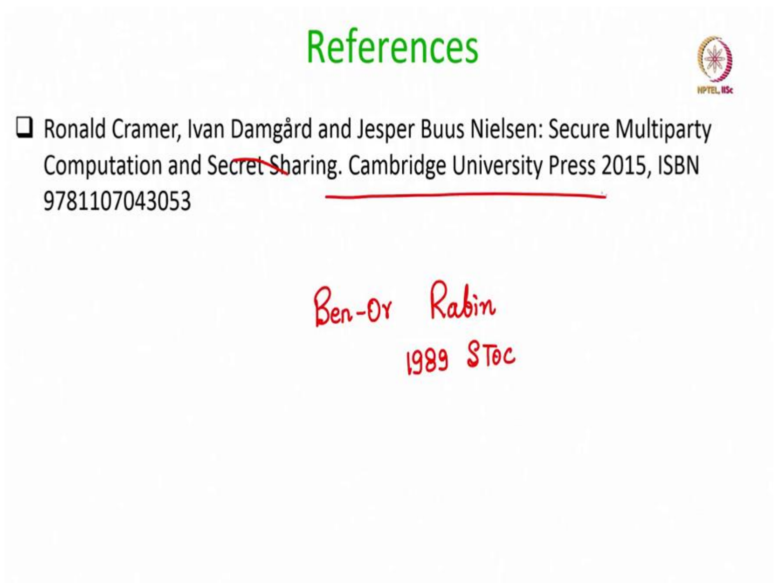
Looking ahead for designing our verifiable secret sharing scheme with statistical security, we will be using information checking protocol which should generate IC signatures with linearity property. So, what does that mean? So, suppose the signed message which intermediary holds after the verification phase is denoted by this notation; that means, signer has given to intermediary a signature on the message; IC-Sig means whatever information INT holds at the end of the verification phase.

Now, the linearity property means here that if a signer has given 2 signed messages to intermediary through two different instances of ICP protocol, then later if INT wants to get signers signature on a linear combination of messages. Then, it should be possible for INT to do that just by performing local operations. And, in the same way if individual verifiers would like to get their corresponding information, corresponding verification information corresponding to this linear combination of message; they should be able to do that locally.

That means, if there are two documents signed by the same signer and given to INT and if in the protocol it is required that there is some linear combination which is publicly known. And, INT needs to get signer's signature on this linear combination of 2 messages; it need not have to run a fresh instance of ICP. INT can generate its part of the signature; well INT can generate its IC signature just by performing linear operation on whatever signatures, IC signatures it has received for m and m' from the signer.

And, correspondingly verifiers also can compute their part of the verification information from the verification information corresponding to m and m', that is what we mean by the linearity property of IC signatures ok. So, that is all about syntax and semantics of IC signatures, IC protocol.

(Refer Slide Time: 27:46)



References

- ❑ Ronald Cramer, Ivan Damgård and Jesper Buus Nielsen: Secure Multiparty Computation and Secret Sharing. Cambridge University Press 2015, ISBN 9781107043053

Ben-Or Rabin

1989 STOC

And, we have not yet seen an exact instantiation of the ICP. In the next lecture, we will see an exact instantiation of the ICP. There are several well-known instantiations of ICP starting from the classic construction by Ben-Or and Rabin. So, that was the starting point of statistically secure MPC.

This was the seminal work by Ben-Or and Rabin in 1989, STOC paper where they formulated the notion of ICP and gave the first construction of ICP. But, the discussion we had in this lecture is taken from this textbook on Perfectly Secure MPC.

Thank you.