6.897 : Selected Topics in Cryptography Profs. Ran Canetti and Ronald Rivest
Lecture 7: The Universal Composability Framework – Definitions
February 26, 2004 *Scribe: Vinod Vaikuntanathan*

1 Review of the previous lectures

In the past lectures, we saw a general framework that can be used to argue about the security of protocols used in multi-party function evaluation. We saw the notion of *modular composition* that allowed us to reason about the security of protocols formed by composing many secure protocols, and we proved the *composition theorem* in the *non-concurrent* setting. Further, we showed how to capture zero- knowledge(ZK) proofs of knowledge (PoK) in this framework, and we analyzed the security of a basic ZK protocol using the composition theorem.

1.1 Review of the definitions

Informally, the security of a protocol was captured by the following condition: We say that a protocol P securely realizes a functionality F if for any adversary A in the real world, there is an ideal-world adversary S, such that no environment can distinguish between a run of the protocol P in the real world and the run with the ideal functionality F and ideal-world adversary S. (For formal definitions, see Lectures 1 and 2).

We know that under this definition of security, there exist protocols for securely evaluating any multi-party function. For example, Goldreich-Micali-Wigderson (GMW) protocol evaluates any n-party function in the presence of a computationally- bounded adversary that controls $t < n$ players. Similarly, BenOr-Goldwasser-Wigderson (BGW) protocol provides "information-theoretic" security against an adversary that corrupts $t < \frac{n}{2}$ players. (i.e, it requires a honest majority). There are a large number of other protocols that work against adversaries of varying power.

2 Motivation for the UC framework

2.1 Characteristics of the basic definition

The basic system model that we saw in the previous section supports only a fixed set of parties, fixed identities for the parties, a fixed number of protocols that can run in the system and a fixed set of corrupted parties. The real world, however, is much more complex. We need a unified framework to analyze security of protocols that takes into account all these complex situations that can arise in the execution of a protocol.

In addition, the basic framework supports only multi-party function evaluation. It does not, in particular, support *reactive* tasks where the output of one phase may be used as an input to the subsequent phases. The security of such a task has to be reasoned based on the collective distribution of the outputs of all the phases. Examples include encryption and signature schemes where the same key is used for processing many messages.

Finally, using the previous definition, we achieve only non-concurrent composition of protocols.

2.2 What do we want from a more general framework ?

A more general framework should achieve the following goals:

1. Deal with more "real-life" settings such as asynchronous communication, unreliable and unauthenticated communication, variable (even unbounded) number of participants and variable identities for the participants.
2. Deal with reactive tasks such as encryption, signatures, commitments and secret-sharing.
3. Deal with an adversary that adaptively corrupts the parties
4. Deal with concurrent composition of protocols. i.e, it should guarantee that security of protocols are preserved under concurrent composition.
5. Allow proving the security of "natural protocols"

The UC framework tries to achieve all the above goals.

3 The UC Framework

The UC framework preserves the overall structure of the following intuitive framework of defining security of protocols: We first formulate a *real-life model*, that represents the protocol execution in the real world. To capture the security requirements of a given task, we next formulate an *ideal process* for the task. We now say that a protocol securely realizes the given task, if running the protocol in the real-world *is in some sense equivalent to* emulating the ideal process for that task.

The UC framework generalizes the underlying computational model ("systems of interacting TMs") and the real-life model of protocol execution. Specifically, the communication in the network is unauthenticated and asynchronous (without guaranteed delivery of messages).

The UC framework generalizes the notion of a "trusted party" by a general entity called the *ideal functionality*, which is modeled as another ITM that repeatedly receives inputs from the parties and replies with the appropriate output values. This generalization is aimed at capturing reactive tasks in the framework.

More importantly, the UC framework generalizes the notion of protocol emulation by introducing an *environment machine* that represents "all the information that is external to the execution of the specific protocol at hand". The environment has arbitrary external input that is known only to itself and it interacts freely with the parties and the adversary [1]. The security requirement is that the protocol execution in the real-world should look the same as the ideal process *from the point of view of any feasible environment.* We also note that the interaction between the environment and the adversary is restricted to be black-box. Jumping ahead, we remark that this is essential to the proof of the composition theorem (which appears in Lecture 8).

3.1 Some Definitions

Next, we present some definitions of the system model.

[1] as opposed to the environment machines that we saw in the previous lectures, which interacted with parties and the adversary only very few times in the course of the computation

- **Interactive Turing Machines (ITM):** An ITM is a TM with some special tapes:
  - An Incoming communication tape, which is read-only.
  - An Incoming subroutine output-tape, which is read-only.
  - *An Identity tape, that contains $ID = (SID, PID)$ where SID is the session ID and PID is the party ID.* [2]
  - The security-parameter tape

  An activation of an ITM is a computation of the ITM until a waiting state is reached.

- A ITM M is a **polytime ITM** if at any time, the overall number of steps taken is polynomial in the security parameter plus the overall input length (the number of bits in the input tape).

A system of interacting ITMs is a pair (M_0, C) where M_0 is the initial ITM and C is the control function from sequences of requests to $\{0, 1\}$. A run of the system is the following process :

- M_0 starts with some external input and a value for the security parameter.
- In each activation, an ITM may request to write to *at most one tape* of another ITM. A request includes
  1. Identity of the requesting ITM
  2. Identity of the target ITM and the code for the target ITM.
  3. Contents of the tape.

  If the control function C allows the tuple (source id, target id, code, tape), then the instruction is carried out. *If no ITM with the target id exists, then a copy is invoked with the said identity, code and security parameter.*
- The machine written to is the next to be activated. If none, then M_0 is activated next.
- The output is the output of the initial TM M_0.

Note 1 *The identity of each ITM is globally unique.*

The following notations are used in our discussion: If an ITM M is invoked because another ITM M' wrote to its input tape, then M is called a subroutine of M' and M' is called the invoker of M.

A state of the system describes an instance in the run of the system, which includes the local states of all the ITMs. A multi-party protocol is a single ITM. An instance of the protocol P with SID *sid* in a state s of the system, is the set of all ITMs in s whose code is P and whose SID is *sid*.

Adversaries and ideal functionalities are also modeled as ITMs, but with the extra power of read/write access to tapes that are part of other ITMs. Another consequence of the definition is that the adversaries are treated as non-uniform TMs, and this accounts for arbitrary external inputs.

[2] The emphasized parts indicate the new components of the definition.

3.2 Models for protocol execution - The real world

The real-life model for executing protocol P with environment Z is the following system of interacting ITMs.

1. Initial ITM: The environment Z is the initial ITM. It has a fixed ID.
2. The control function C specifies the following.
   - Z can activate a single copy of an ITM A (the adversary) and multiple ITMs running P, all having the same SID, and write to their input tapes.
   - A can write to the incoming communication tapes of all the parties and to the subroutine output-tape of Z.
   - All the other ITMs can can write to the incoming communication tape of A, can invoke new ITMs, and can write to the subroutine output-tape of of their invoker and the input tapes of their subroutines.
   - **Modeling corruptions:** A can write a "corrupt" message on the incoming communication tape of ITM M. Then, M writes "corrupted" on the subroutine output-tape of Z. From now on, in each activation, M sends its entire state to A. A assumes all write privileges of M.

The environment Z interacts with A freely throughout the computation. It is easy to see that no generality is lost by assuming that A reveals its entire internal state to Z. But the interesting cases of protocol executions occur when Z holds some additional information that is secret to A, and tests if this secret information and the information that A outputs are correlated. Thus, we cannot, in general, assume that Z reveals its entire state to A.

All communication between parties is done via A. This, in effect, models open, unauthenticated channels. The communication is asynchronous with no authenticity or reliability guarantee. Z creates new parties adaptively and sets their identities. In particular, we model adaptive corruptions.

Let $\mathsf{EXEC}_{P,A,Z}(k, z, r)$ denote the output of Z after above interaction with P, A, on input z and randomness r for the parties with security parameter k. Here, r denotes the concatenation of randomness of all the parties. $\mathsf{EXEC}_{P,A,Z}(k, z)$ denotes the output distribution of Z after above interaction with P, A, on input z and security parameter k, and uniformly chosen randomness for the parties. $\mathsf{EXEC}_{P,A,Z}$ denotes the ensemble of distributions $\{\mathsf{EXEC}_{P,A,Z}(k, z)\}_{k\in N, z\in\{0,1\}^*}$.

3.3 Models for Protocol Execution - Ideal Process, Ideal functionalities and Dummy parties

An ideal functionality is a PPT ITM F with the following properties:

1. The PID of F is unused (say, set to 0).
2. F takes inputs from multiple parties. The only restriction is that it takes input only from parties whose SID is identical to the local one.
3. F can write outputs to all parties that write inputs to it, and invoke new dummy parties with the same SID.

Dummy parties are very simple ITMs that forward their inputs to the functionality, and output whatever the functionality outputs. More precisely, the dummy party for F does the following:

1. When activated, it copies all its inputs to the incoming communication tape of the functionality F.
2. Copy all outputs from F to the subroutine output-tape of its invoker.

The ideal process for evaluating functionality F with environment Z and adversary S is the following system of interacting ITMs:

- Initial ITM: Environment Z (the initial ITM, with fixed ID)
- Control function C specifies the following: Z can activate a (single copy of) an ITM A (the adversary), and multiple copies of ITMs running P, and write to their input tapes. However, here A is "ideally replaced" with an ITM S, and the parties running P are "ideally replaced" by dummy parties for F. F can write to the communication tape of S and to the subroutine output tapes of all dummy parties. S can write to the incoming communication tape of F and to the subroutine output-tape of Z.
- **Modeling corruptions:** A can write a "corrupt M" message on incoming communication tape of F. Then, F will write "Corrupted" on subroutine output tape of M, reveal some information to S and let S influence the output that F provides to M.

Note: Communication from Z to F and back, via the dummy parties, is immediate. Also note that the functionality F knows who is corrupted. The allowed information leakage upon corruption has to be explicitly specified by talking about a class of adversaries.

Let $\mathsf{IDEAL}^f_{S,Z}(k,z,r)$ denote the output of Z after above interaction with F and S, on input z and randomness r for the parties with security parameter k. Here, r denotes randomness for all parties. Notationally, $r = (r_Z, r_S, r_f)$. Let $\mathsf{IDEAL}^f_{S,Z}(k,z)$ denote the output distribution of Z after above interaction with f and S, on input z and security parameter k, and uniform randomness for the parties. $\mathsf{IDEAL}^f_{S,Z}$ denotes the ensemble $\{\mathsf{IDEAL}^f_{S,Z}(k,z)\}_{k\in N, z\in\{0,1\}^*}$.

3.4 Definition of Security

Definition 1 *Let C be some class of real-life adversaries. We say that a protocol P securely realizes the ideal process for F* with respect to C *if for any adversary $A \in C$ there exists an adversary S such that for all environments Z we have that*

$$IDEAL^F_{S,Z} \sim EXEC_{P,A,Z}$$

The ideal process is not required to generate an output. In the most general setting, where message delivery is not guaranteed, it is impossible to impose a requirement that the protocol terminates. Instead, we say that the security requirements are met *in the case that the protocol generates output.*

Note that the above definition meets the intuitive requirement that the security of a session running in some host should be as independent as possible on the security of other sessions on the same host. As a result, the definition guarantees security on a session-by-session basis.

Also note that there is no parameterization of the security via the "set of corrupted parties". We stress that in the current formulation there is no need for this, since the functionality F knows who is corrupted. Now, the security properties of the protocol under corruptions can be explicitly expressed in the code of F.

3.5 Some Variants

The general definition works in the case of active adversaries, i.e, adversaries that have total control over the behavior of the corrupted parties. In the case of passive (semi-honest) adversaries, the corrupted parties continue running the original protocol. This can be easily modeled by making the following changes: In the real-world, even after being corrupted by the adversary A, an ITM that corresponds to the corrupted party keeps getting activated and generates messages and outputs. The only difference is that now, A has read access to the memory contents of the corrupted ITM. Alternatively, we can require that the corrupted party forward its entire state to A in each activation. A similar change is made to the behavior of the corrupted party in the ideal world. Also, in the ideal world, the adversary S cannot send messages to F with the ID of the corrupted party.

To model unconditional security, we allow Z and A to be computationally unbounded. We stress that the running time of S should remain polynomial. Perfect security can be modeled by saying The outputs of Z in the two runs should be identically distributed. Other variants such as secure channels, authenticated channels, and synchronous communication are captured as ideal functionalities within the existing framework, without changing the framework itself.

3.6 Some equivalent formulations

1. Z outputs an arbitrary string (rather than one bit) and the outputs of Z in the two executions should be computationally indistinguishable. Suppose there is a Z that outputs a large string and that
$$\mathrm{IDEAL}^{F}_{S,Z} \not\approx \mathrm{EXEC}_{P,A,Z}$$
Then we can construct a Z' armed with Z and a distinguisher for the two distributions $\mathrm{IDEAL}^{F}_{S,Z}$ and $\mathrm{EXEC}_{P,A,Z}$. The new environment outputs a bit and by definition of Z, successfully distinguishes between the real run and the ideal run. Note that this argument is valid because we universally quantify over all environments.

2. Z and A are limited to be deterministic. We allow the environment to receive arbitrary input. Thus, we consider adversaries and environments as non-uniform ITMs. Since non-uniformity is at least as powerful as randomness, we can without loss of generality, restrict A and Z to be deterministic.

3. Changing the order of quantifiers in Definition 1. i.e, Now we let S depend on Z. The formulations are actually equivalent. [3]

In the next lecture, we will look at another equivalent formulation and go on to state and prove the UC theorem.

[3] We will not give a proof of this in class.