ECE598 Ideal Functionalities, Spring 2022 Lec 03: Recap UC Security and Introduction to ITMs

Lecturer: Andrew Miller
Scriber: Peiyao Sheng (psheng2)

Date: January 25, 2022

1 Overview

This lecture first analyzes commitment protocol in the random oracle model, which requires the behavior of the dummy adversary in the real world to be captured by a simulator in the ideal world. The second part of the lecture introduces the composition theorem, extends protocols to multi-sessions, and briefly introduces the ITMs used in SaUCy.

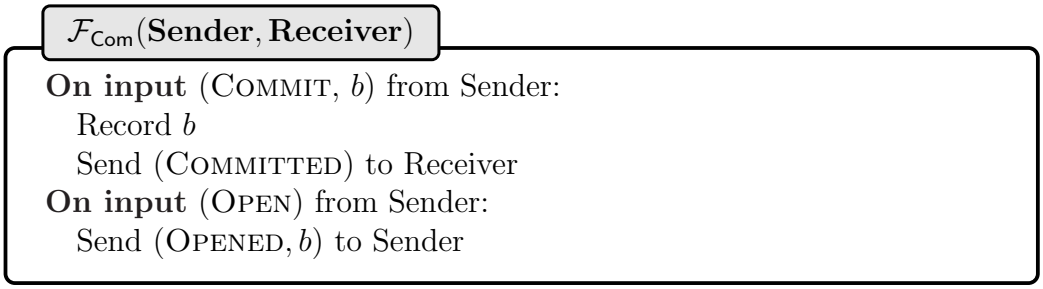

Figure 1: Commitment Ideal Functionality

2 UC Security for Commitment Protocol

A protocol π is said to securely realize a given ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ (or for dummy adversary by dummy lemma) there exists a simulator $\mathcal{S}$ in ideal world, such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol in real world, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in ideal world [1]. For the commitment protocol under the random oracle model (introduced in the previous lecture) to achieve $\mathcal{F}_{\mathsf{Com}}$ in Fig 1, the requirements of ideal-world simulator are discussed in two cases. First, in the real world when the receiver is corrupt, the simulator $\mathcal{S}$ must be able to generate a simulated commitment that can be opened to any value. In the second case, when the sender is corrupt, $\mathcal{S}$ has to extract the committed value from the commitment.

2.1 Case 1: Receiver is Corrupt

Real world. In the real world of the first case, sender is controlled by an honest party, when the environment $\mathcal{Z}$ activates the sender to create a commitment for b, the sender will generate a random nonce r and query $\mathcal{F}_{\mathsf{RO}}$ for the hash of $b\|r$ which replies it with a commitment h. The Committed together with h will be sent to the receiver (controlled by a dummy adversary) through $\mathcal{F}_{\mathsf{2way}}$ (two-way communication ideal functionality, which is assumed to be used in every communication). When the sender is asked to open the commitment, it will send $(\text{Opening}, b, r)$ to the receiver, which will be forwarded to $\mathcal{Z}$. In addition, $\mathcal{Z}$ can ask $\mathcal{S}$ to query $\mathcal{F}_{\mathsf{RO}}$, and it should return the result of random oracle query.

Ideal world. In the ideal world, the Commit instruction will be sent to $\mathcal{F}_{\mathsf{Com}}$, then $\mathcal{S}$ will receive Committed but without payload. And $\mathcal{S}$ must be able to construct some hash for the commitment. Similarly if receiving (Opened, b) from $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{S}$ must be able to show $(\text{Opening}, b, r)$ to $\mathcal{Z}$.

The first attempt to construct a simulator is to sample everything. And we will show how such a simulator mess things up. We suppose $\mathcal{S}$ interacts with $\mathcal{Z}$ as follows:

- On $(\text{RO-Query}, m)$: Randomly sample h and store (m, h) if m has not yet been stored. Otherwise return h for stored (m, h).
- On Committed: Randomly sample h and return $(\text{Committed}, h)$.

- On $(\textsc{Opened}, b)$: Randomly sample r, return $(\textsc{Opening}, b, r)$.

Distinguishability. The problem of the above simulator is that it fails to ensure the consistency of random oracle queries and the opening results. Thus an environment can first ask the sender or $\mathcal{F}_{\mathsf{Com}}$ to commit 0 and get h, then ask them to open the commitment and get (b, r). Later, if the response from querying RO for $b\|r$ is h', $\mathcal{Z}$ can simply check whether $h = h'$. While in real world they are always equal since the honest sender knows r, in ideal world, it is highly likely that two hashes are not consistent, according to which $\mathcal{Z}$ can easily distinguish two worlds.

Fix the simulator. To fix the above problem, the simulator needs to store $(b\|r, h)$ in the random oracle table, where h is the previously sampled hash in response to $\textsc{Committed}$. In this case when $\mathcal{Z}$ query the oracle with the opening tuple, it will receive the same h. To complete the proof we need to consider all the other possible behaviors of the environment, which include (1) input to the honest sender to commit b and expect to receive h, (2) query any random oracles, (3) ask the honest sender to open and expect ot receive (b, r), (4) query more random oracles. For all these cases, the simulator can always simulate the real world except that $(b\|r, h)$ has already been queried, which happens with negligible probability since the environment can only query with polynomial time.

2.2 Case 2: Sender is Corrupt

Real World. When the sender is corrupt, as the opposite of the first case, the environment will generate the commitment h based on secretly sampled b and r, and generate $(\textsc{Opening}, b, r)$ later to open the commitment.

Ideal World. In order to simulate the behavior of corrupted sender, $\mathcal{S}$ must be able to provide the ideal functionality with a value for the committed bit on receiving the commitment from $\mathcal{Z}$. In other words, $\mathcal{S}$ has to "extract" the committed bit from the commitment generated by $\mathcal{Z}$. Notice that the extraction must be done before the open request, at which time $\mathcal{S}$ has no extra information about b except h.

Extractability. Though the "hiding" property of the commitment protocol ensures that no information about b will be revealed from h, the simulator in ideal world also simulates the random oracle. And any valid commitment h must be generated by querying the random oracle and the associated b, r must be stored in the random oracle table. Thus the simulator can easily extract b from the table given h. Later when receiving $(\text{OPENING}, b, r)$ from $\mathcal{Z}$, the simulator needs to check the table to see whether it matches the existing queries.

3 Composition Framework

So far we have talked about the real-ideal paradigm and the commitment protocol as an example. In this section we will formally define the framework and notations used in the above example.

Protocol and Ideal Functionality. In Section 2, we discuss whether the commitment protocol π_{comm} realizes commitment functionality $\mathcal{F}_{\mathsf{com}}$ in the random oracle model. We define the following notation

$$\mathcal{F}_1 \xrightarrow{\pi} \mathcal{F}_2$$

to represent protocol π realizes functionality $\mathcal{F}_2$ and relies on functionality $\mathcal{F}_1$. In other words, for dummy adversary, there exists a simulator $\mathcal{S}$, such that for all environment $\mathcal{Z}$,

$$\mathsf{Real}_{\mathcal{Z},\pi,id,\mathcal{F}_1} \sim \mathsf{Ideal}_{\mathcal{Z},id,\mathcal{S},\mathcal{F}_2}$$

where $\sim$ means computationally indistinguishable.

3.1 Composition Theorem

The protocols can be composed by interacting among components. We call $\varphi \circ \pi$ the "composed protocol" of φ, π if $\varphi \circ \pi$ is identical to φ with the exception that each interaction with some copy of $\mathcal{F}$ is replaced with a call to an appropriate instance of π (by directly forwarding the messages in $\mathsf{p2f}$ channel of φ to the $\mathsf{z2p}$ channel of π, where $\mathsf{p2f}$ means protocol-to-functionality, $\mathsf{z2p}$ means environment-to-protocol), and the outputs of π are treated as inputs to $\mathcal{F}$. With these notations, the composition theorem can be described as follows,

Theorem 1 *(Composition theorem) For any functionalities $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$ and protocols π, φ, if we have $\mathcal{F}_1 \xrightarrow{\pi} \mathcal{F}_2$ and $\mathcal{F}_2 \xrightarrow{\varphi} \mathcal{F}_3$, then*

$$\mathcal{F}_1 \xrightarrow{\varphi \circ \pi} \mathcal{F}_3$$

The theorem can be proved by contraposition. Naturally, in ideal world, the simulators of composed protocols will also compose and once the environment can distinguish the composed simulator, at least one of the simulators of original protocols can be distinguished by the environment.

3.2 Universal Composition

Multisession Extension. In general, a single shot functionality can be extended to multiple sub-instances, we use $!\mathcal{F}$ to denote the multi-session extension of $\mathcal{F}$. Each sub-instance of $\mathcal{F}$ has an associated identifier called ssid, i.e. sub-session identifier. Specifically, a new instance will be created at the first time its ssid is called. We formulate the natural extension of $\mathcal{F}_{\mathsf{Com}}$ that handles multiple commitment requests and call it $\mathcal{F}_{\mathsf{Mcom}}$ (in Figure 2). Combining composition theorem and multisession extension, we get the universal composition theorem stated below

Theorem 2 *(Universal Composition) For any functionalities $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$ and protocols π, φ, if we have $!\mathcal{F}_1 \xrightarrow{\pi} \mathcal{F}_2$ and $!\mathcal{F}_2 \xrightarrow{\varphi} \mathcal{F}_3$, then*

$$!\mathcal{F}_1 \xrightarrow{\varphi \circ \pi} \mathcal{F}_3$$

The theorem indicates that if we have multiple instances of protocol π (and φ), which can realize a single instance of $\mathcal{F}_2$ ($\mathcal{F}_3$) using multiple instances of $\mathcal{F}_1$ ($\mathcal{F}_2$), then we can use multiple instances of the composed protocol $\varphi \circ \pi$ to realize a single instance of $\mathcal{F}_3$ by calling multiple instances of $\mathcal{F}_1$. The proof of the theorem will be discussed in future lectures.

4 Introduction to ITMs in SaUCy

ITMs as a Process Calculus Everything we have discussed so far is able to be simulated by a system of interactive Turing machines (ITMs), where each ITM represents the program to be run within a different party. We use channels to conduct send/receive operations of concurrent processes

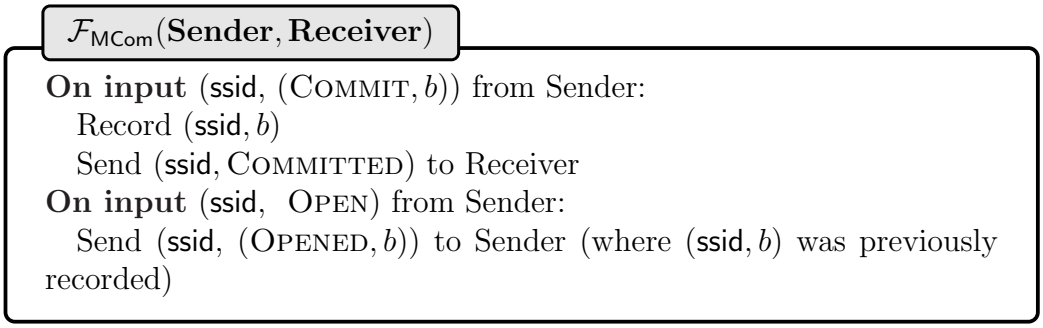

Figure 2: Multi-session Commitment Ideal Functionality

(by write/read of channels). At the beginning of the protocol, new channels need to be created, and different components of the system interact with each other via different channels. To apply reduction for proofs, every execution of any system needs to be Probabilistic Polynomial Time (PPT), which can be ensured by following the ITM channel rules:

1. No duplication of "read" ends of channels.
2. No parallel composition of "write" active process
3. No sending channels over net/io channels

The first two rules ensure that there is no race condition resulting from multiple reads or writes to a single channel.

References

[1] R. Canetti and M. Fischlin. Universally composable commitments. In *Annual International Cryptology Conference*, pages 19–40. Springer, 2001.