ECE598 Ideal Functionalities, Spring 2022
Lec 04: Universal Composition, Why Composition is hard

Lecturer: Andrew Miller
Scriber: Bolton Bailey (boltonb2)

Date: January 27, 2022

1 Overview

Previously in this course, we were introduced to the Universal Composability framework. We've gained an understanding of the 4 roles that play a part in a UC proof, how they interact in the ideal and real worlds, and how the dummy lemma allows us to simplify these proofs. Last time, we took a first look at composition, seeing how interactive Turing machines work, and stating the composition theorem.

Today, we look at some of the pitfalls with composition. In particular, we look at the difference between parallel and sequential composition, and how some protocols which are secure in the latter setting and not secure in the former. We make this explicit by describing a protocol that can leak a secret of one of the parties when run in parallel.

We then take a look at Pedersen commitments in the UC setting. We discuss why the conventional Pedersen commitment is secure against a corrupted receiver but not a corrupted sender. We mention some details of Blazy et al. [1], which describes a way of getting around this, to produce a UC-secure commitment in the CRS setting.

2 ITM Review

Interactive Turing Machines (ITMs) are used to model of the interactions and computations of parties in UC. We would like to model concurrency, but we need the execution of any system to be probabilistic polynomial time (PPT). This can be achieved by following the three rules for ITM channels:

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

For example, if we wanted to model a broadcast channel, we could do so using a for loop, as in Figure 1.

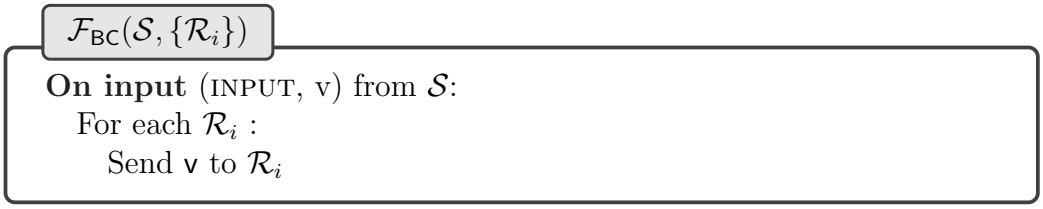

Figure 1: An example of a bugged broadcast functionality incompatible with the ITM model.

The issue with this naive construction is that when the message is sent to the first $\mathcal{R}_i$, there is no way for $\mathcal{S}$ to reclaim control of execution by reading anything. Since only one process can be executing at a time, and processes are only woken up by reading, this implementation will get stuck. We can't just solve the problem by executing the sends "in parallel", since we don't want to create race conditions. A better implementation gets around this by having the protocol save the sent message, and letting the $\mathcal{R}_i$ parties query the protocol for the message later, see Figure 2 (although this is still not perfect, since in a real protocol, there would be network delays, whereas in this version, the parties get immediate access to the value, but we'll go into this more later when looking at consensus protocols).

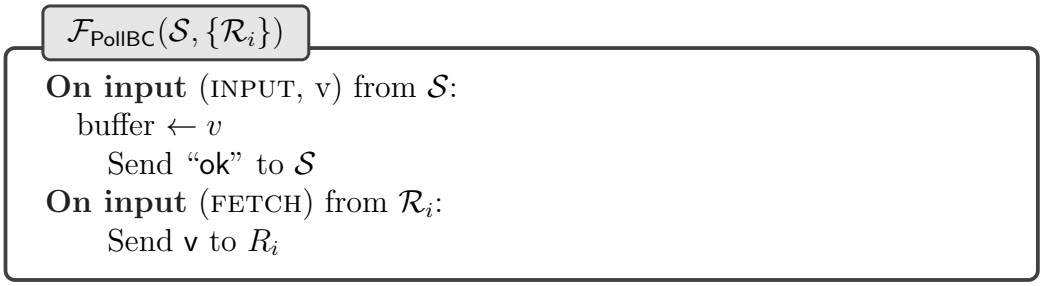

Figure 2: An example of a broadcast functionality that works in the ITM model using polling.

3 Concurrent Composition

The essential problem in this section has to do with the difference between sequential and parallel composition.

- Sequential Composition is when a protocol is used multiple times, but each iteration of the protocol only starts after the previous one has completely finished.
- Parallel composition lets multiple instances of a protocol run at the same time.
- There's also arbitrary concurrent composition, which specifically allows for the adversary to run steps of a protocol in any order they want, but we'll treat this as a form of parallel composition.

Imagine that a prover and verifier are using a Two-Party interactive proof protocol multiple times. Externally, these two approaches look the same - the verifier takes some values for which it requests proofs. The prover receives as input witnesses that allow it to complete those proofs. The verifier then might accept or reject. From a practical point of view, parallel composition seems better, just because it might mean less wall-clock time for the protocol as a whole to complete.

The issue comes when information that appears in one instance of the protocol lets a party do something in another instance of the protocol that they would not otherwise be able to do.

The following protocol from Goldreich and Krawczyk [2] doesn't output any data, but still illustrates this concept of how a secret can be leaked through parallelism. The protocol assumes the existence of some one-way function family f, which can be evaluated using a key k, (i.e. $f_k(AB) = \mathsf{OWF}(k||A||B)$. There are two parties: P, who is given some key k with which to evaluate f and who also has some secret s, which the other adversarial party A is not supposed to learn.

The protocol has two modes, and the adversary A can choose which to execute. In the first mode, party P selects a uniformly random n-bit string α, and sends it to party A. Party A replies with two n-bit strings β, γ (selected however it wants, since it's an adversary). P checks if $f_k(\alpha||\beta) = \gamma$, and if this is the case, leaks the secret s to A.

In the second mode, A selects α and sends it to P. P selects β uniformly at random and replies with $\beta, f_k(\alpha\beta)$.

Notice that with a single instance of the protocol, A will never be able to learn s. The only mode where s could possibly be revealed is the first mode, but to cause A to reveal s in this case, the adversary would have to generate an evaluation of the function f, which it can't do without learning k.

Furthermore, this protocol cannot be broken under a sequential composition model. Supposing the adversary has access to transcripts from previous runs of the protocol, she would like to produce an evaluation of f for the α given by P and arbitrary β. But since α was generated randomly by the prover at the start of this round, it is negligibly likely that this α matches any α from a previous round, whether that previous round was in the first or second mode.

But in the arbitrary concurrent parallel composition model, the secret can be leaked. The key is for the adversary to use one instance of the second mode in the middle of the execution of the first mode. It works like this:

1. A starts an instance in mode 1. P generates α and sends this to A
2. A now starts a second instance in mode 2, and sends this α value as its first message.
3. P in the second instance now replies with a random β and the evaluation $\gamma := f_k(\alpha\beta)$.
4. A now uses this β, γ pair as it's response in the first instance.

5. P checks this β, γ pair and finds that, indeed, $\gamma = f_k(\alpha\beta)$. It leaks the secret s to A

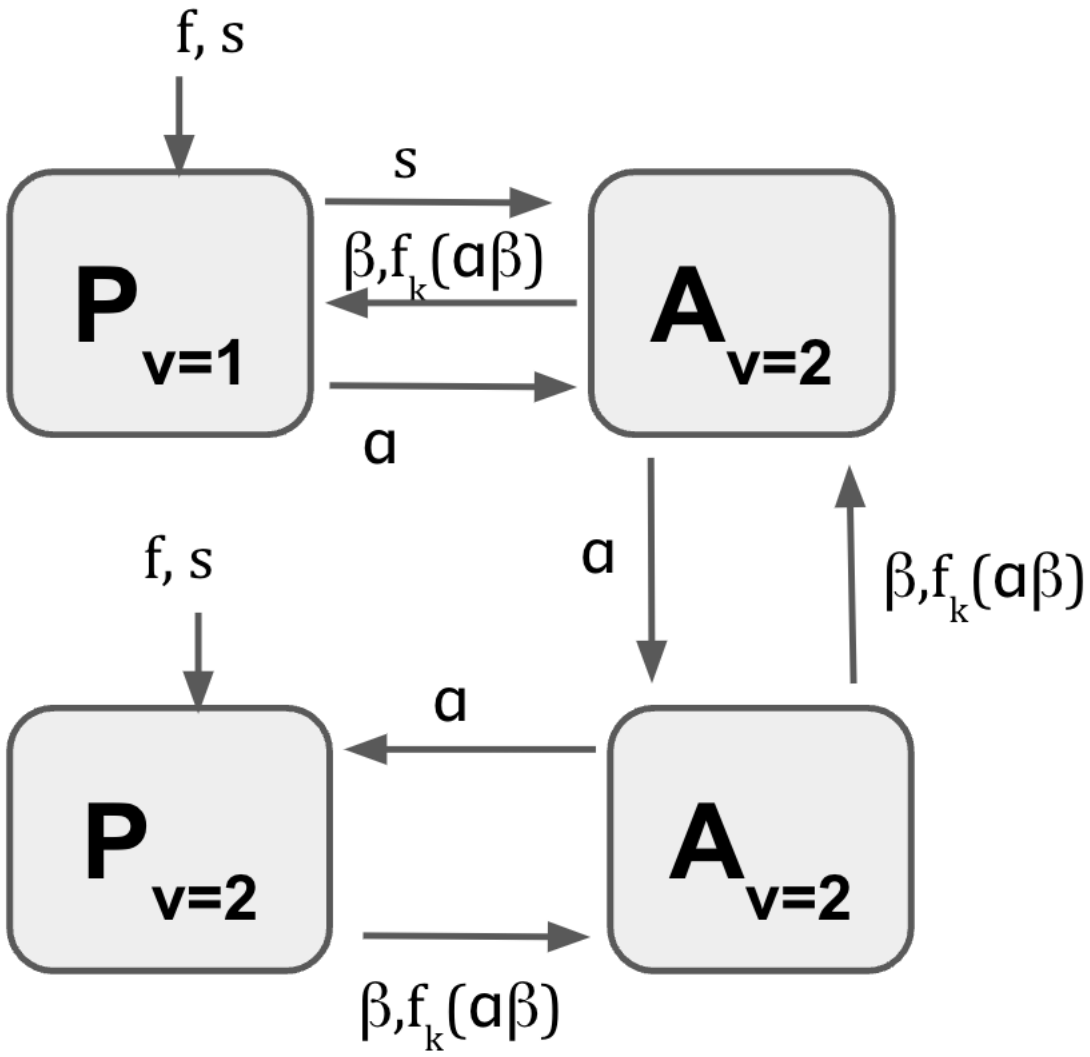

Figure 3: A depiction of the information flow through the two modes as the adversary attacks the protocol and learns the secret.

This isn't exactly a UC Protocol, since it's not clear what functionality

this implements, but it justifies why UC proves its protocols to be secure in the arbitrary concurrent composition model.

Related to the Goldreich and Krawczyk paper is [3], which shows that Byzantine Agreement is not secure under secure under concurrent composition, depending on how messages are authenticated.

4 Are Pedersen Commitments UC secure?

Recall the Pedersen Commitment scheme: There is a setup phase where we have two elements sampled at random from a cryptographic group $g, h \xleftarrow{\$} G$. One then commits to a value x by sampling a random $r \xleftarrow{\$} \mathbb{Z}_p$ and outputting $C = g^x h^r$. This protocol has the advantage of removing the random oracle model from commitment.

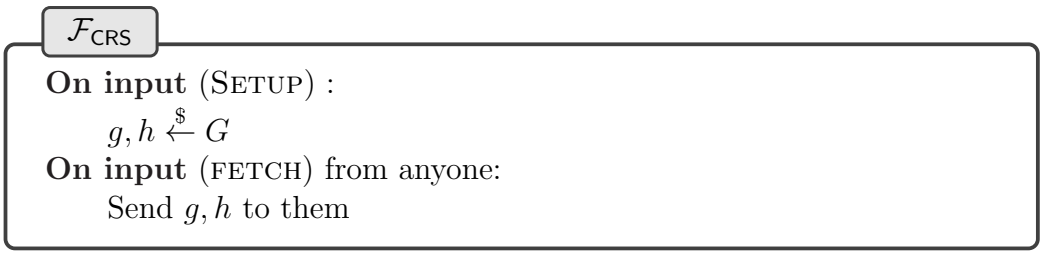

Figure 4: A simple model for CRS in UC.

The hiding property of this scheme is information-theoretically perfect. Since r is random, the commitment C is uniformly distributed over all possible group elements. The binding property follows by reduction to the Discrete Log problem: You can show that if one can find a collision, one can compute the discrete log between g and h. One could write out the receiver and committer protocols as follows.

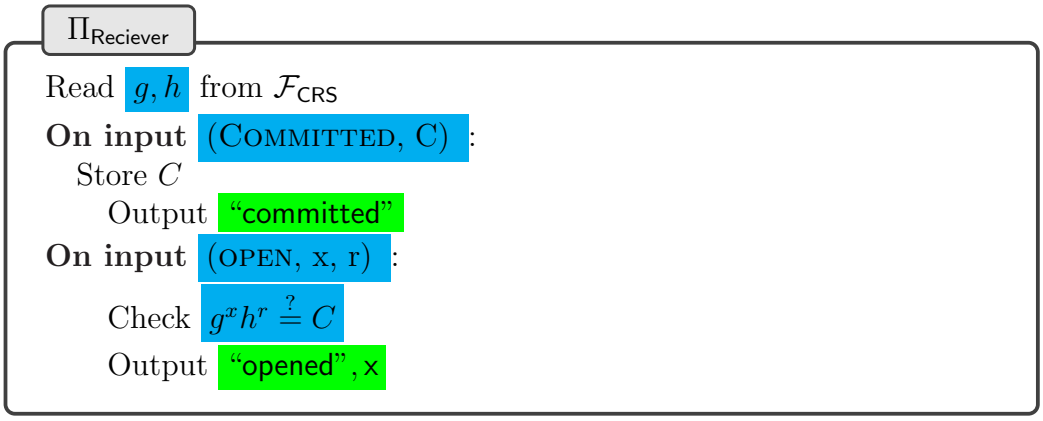

Figure 5: A model for the receiver. Defined by $\mathcal{F}_{\text{Com}}$ are in green, and choices from the protocol are in blue

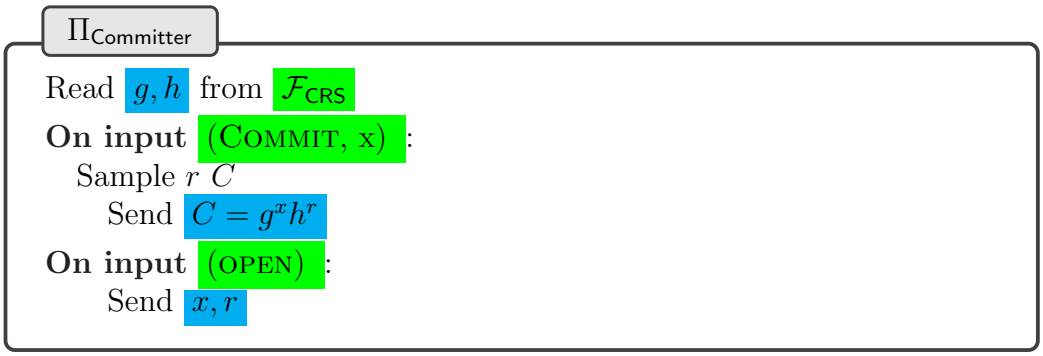

Figure 6: A model for the committer.

It is in fact possible to carry through the UC proof for the case where the receiver is corrupt. When the receiver is corrupt, it receives committed messages from $\mathcal{F}_{\text{Com}}$, but these carry no information, and the receiver has to show a group element to the environment. The simulator, then, just has to provide the value before receiving the message bit. It can do this by generating a trapdoored CRS, $g, h = g^\tau$. When sent the "committed" message, it can pass $C = g^m$ through to the environment. Later, when sent an "open" message for bit b, it solves $m = b + \tau r$ and outputs b, r to the environment. Thus, this is stronger than hiding, it is actually *equivocable*.

The issue comes for the proof of UC security against an adversarial sender. When the sender is corrupt, the message flows the other way, the way the simulator gets activated is instead by the environment sending a protocol message (because the values sent by the environment are instructions to dummy adversary to control the corrupt sender). So the simulator gets activated with a C, and the simulator must activate the ideal functionality. But this gets stuck; the simulator can still learn the trap door, but it gets a uniformly random C, and there is no information about the bit to be found in C and the trapdoor doesn't help.

Solving this problem, by making a UC-secure version of this commitment, is the programming assignment for the course. As a reference Blazy et al. [1] is helpful, which builds off of Lindell et al. [3]. The idea is to augment the commitment with an encryption of the bit, so that the simulator can learn the bit, as well as a zero-knowledge proof to guarantee the encryption is correct.

References

[1] O. Blazy, C. Chevalier, D. Pointcheval, and D. Vergnaud. Analysis and improvement of lindell's uc-secure commitment schemes. In *IACR Cryptol. ePrint Arch.*, 2013.

[2] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25:169–192, 1996.

[3] Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated byzantine agreement. *J. ACM*, 53(6):881–917, nov 2006.