6.897 **Special Topics in Cryptography** Instructors: Ran Canetti

Lecture 9: UC Commitments and Other Feasibility Results

March 4, 2004 *Scribe: Susan Hohenberger*

1 Summary of Previous Topics

In the past few lectures, we have turned our attention to analyzing the security of protocols. We studied two different frameworks for discussing the security of a protocol: the basic and the UC (universal composition) framework.

The basic framework makes certain assumptions about the world in order to keep things as simple as possible. These assumptions are not always realistic (hence the need for the stronger, UC framework), but they offer a starting point for the protocol designer. Any protocol secure in the UC framework is also secure in the basic one, but the converse is not necessarily true.

There are four main assumptions that we made about the world in the basic framework:

1. **Non-reactive Tasks.** We focus only on function evaluation, meaning the set of protocols which require only a single round of inputs and outputs for each party involved.

2. **Synchronous Communication.** Although there are many different types of synchrony, here we define synchronous communication *by the round*. Let there be a sequence of discrete rounds such that at the beginning of every round messages are collected and at the end of the same round they are all delivered. There is no guarantee on the order of delivery for messages *in the same round*, but all messages collected in round i are guaranteed to be delivered before any message is collected in round $i+1$.

3. **Non-adaptive Corruptions.** In the non-adaptive setting, the adversary must indicate, at the start of the protocol, which parties it wishes to corrupt. These parties will remain corrupted throughout the protocol.

4. **Non-concurrent Modular Composition.** We consider only the security threats that arrive from running multiple protocols (either different protocols or multiple copies of the same one) in a serial fashion, meaning that each protocol p_i must halt before protocol p_{i+1} may begin executing.

Since the assumptions made above cannot typically be depended upon in the real world, we add the following four properties to the basic framework so that we can more accurately talk about the secure composition of protocols. We call this new framework the UC (universal composition) framework.

1. **Reactive Tasks.** We focus on general protocols, allowing multiple rounds of inputs and outputs for each party involved.

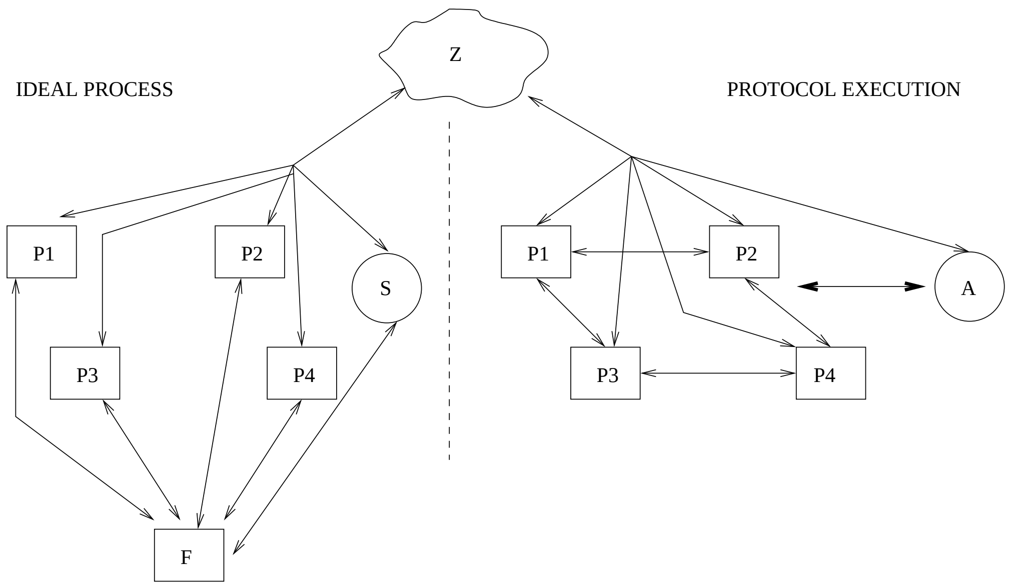

Figure 1: Pictorial view of the definition of security for a protocol π.

2. **Asynchronous Communication.** Generally speaking, there is no synchrony guarantee whatsoever (although there are varying levels of weak synchrony that can also be considered). There is no guarantee on the order in which messages will be delivered, or even that they will be delivered at all.

3. **Adaptive Corruptions.** In the adaptive setting, the adversary may observe the protocol execution for a time before deciding which parties it wishes to corrupt. The adversary is also allowed to uncorrupt parties.

4. **Concurrent Modular Composition.** We consider the real life security threats that arrive from running multiple protocols (either different protocols or multiple copies of the same one) in a concurrent fashion, meaning that protocol p_{i+1} may begin executing at any time relative to the start of protocol p_i. This is generally referred to as "universal composition".

The UC framework captures *most* other composition models; for our purposes, we will consider it "general enough" to model the real world.

Recall the basic diagram for discussing protocol security in Figure 1.

Definition 1.1 *Protocol π securely realizes a functionality F if for any adversary A there exists a simulator S such that no environment Z can tell whether it interacts with:*
- *A* real *run of π with A.*
- *An* ideal *run with F and S.*

Figure 1 and definition 1.1 take on an entirely new character when we consider them to be operating in the UC framework. Since it is a reactive framework, the environment

Z is allowed to talk to all parties throughout the computation (– including the adversary A!). Thus, the adversary can collude with the environment in an effort to cheat the honest parties. Letting the adversary talk freely with Z is an independent definitional decision. This is what allows for concurrent composition. Another way to think of this new framework is to imagine that all messages Z wishes to send to the honest parties (and vice versa) must be relayed through A. Modelling the reactive interaction between Z, A, and the honest parties allows us to talk about the security of concurrently running protocols (e.g., recall from last lecture that we do so by considering dummy adversaries in the hybrid model). Everything hinges on the free communication between Z and A; if their communication is restricted then not only would the current proof of the UC theorem be invalid, the theorem itself would be false!

2 Feasibility Results in the UC Framework

After putting effort into understanding an abstract, UC framework that (hopefully) models the real world, there are some natural questions to ask about it.

How do we write ideal functionalities? The first hurdle to designing protocols in the UC framework is to understand which properties are desired from the *ideal* functionality. This is not always trivial. In lecture 8, we saw two ideal functionalities for key-exchange. In the first example, after receiving "exchange" requests from two parties, the functionality selected a random value and sent it to both parties. However, after some additional thought, we realized that such a functionality is a little *too* ideal. For honest parties, we would expect such behavior from a key-exchange function, but we require no such guarantee when one of the parties is corrupted. We changed the ideal functionality for key-exchange accordingly, only requiring successful key exchange when both parties are honest and letting the adversary choose the key otherwise. We will see more examples of writing ideal functionalities later in this lecture.

Are known protocols UC-secure? Once we understand what we expect out of the ideal functionality, the next hurdle is constructing a real-world protocol that acheives the same goal. We already have several interesting protocols for key-exchange, commitments, encryption, etc., which meet these ideal functionalities and are provably secure in the basic framework, so a natural question to ask is: are the known protocols also UC-secure?

The answer is mixed. We'll start with some good news.

Theorem 2.1 *Multiparty protocols with authentication and an honest majority can realize any functionality. [Can01]*

In fact, the protocols with an honest majority from the basic framework work (with only minor alternations) in the UC framework as well. For example, we can consider the [BOGW88] protocols for up to $\frac{1}{3}$ corrupted players, the [RBO89] protocols for up to any corrupted minority, and the [CFGN96] adaptively-secure protocols for any honest majority using open channels.

However, we are not so fortunate in the two-party case, where there is no honest majority. For two-party functionalities, the known protocols do not work. The security proofs for these protocols often fall apart because they can no longer count on black-box simulation with rewinding – the simulator cannot rewind the UC environment Z. In fact, many interesting functionalities[1] (e.g., commitments, zero-knowledge, coin tossing, etc.) *cannot* be realized in the plain UC model! Today, we will prove this impossibility result for commitments. Does it follow that we can't achieve secure commitment schemes in the real world?

How do we design UC-secure protocols? Fortunately, we can achieve commitments and all the other interesting two-party functionalities in the UC framework by adding a common random string to our model. (We will prove this in a later lecture.)

Theorem 2.2 *UC-secure commitment schemes exist in the common random string model. [CF01, CLOS02, DN02, DG03, HMQ04]*

Theorem 2.3 *UC-secure, non-malleable zero-knowledge proofs exist in the common random string model. [CF01, DCO$^+$01]*

Theorem 2.4 *UC-secure protocols exist for any two-party functionality in the common random string model. [CDN01, CLOS02]*

Theorem 2.4 generalizes to any multiparty functionality with any number of faults.

UC Encryption and Signatures Encryption and signatures are two of the most commonly used cryptographic protocols, and it is very reassuring to discover that our security definitions in the basic framework come very close to realizing UC encryption and do, in fact, realize UC signatures even against an adaptive adversary:

We can write a *digital signature functionality*, F_{sig}, such that realizing F_{sig} is equivalent to security against *adaptive chosen message attack* as defined in [GMR88]. In the ideal functionality for F_{sig}, we keep the secret key of the signer secret from Z. We can use F_{sig} to realize many desirable functionalities in the UC framework, including *ideal certification authorities* and *ideally authenticated communication.*

It is also possible to write an ideal *public key encryption functionality*, F_{pke}, such that realizing F_{pke} against *non-adaptive* adversaries is equivalent to security against *adaptive chosen ciphertext attack* (CCA2) as defined in [RS91, DDN91]. In the ideal functionality for F_{pke}, we keep the decryption key of the each party secret from F. Known CCA2-secure cryptosystems only realize a relaxed variant of F_{pke} and are only UC-secure against non-adaptive adversaries. In fact, this a tricky subject. It is provably impossible to realize F_{pke} with respect to adaptive adversaries in the plain UC framework, but some complicated variants capturing almost all of F_{pke} are possible (see [CHK04]). It is an open problem whether or not a simple implementation of an almost-complete F_{pke} functionality is possible. Another question open for debate is: are F_{sig} and F_{pke} (as will be defined later in the course) really the natural functionalities?

UC Key-exchange and Secure Channels In lecture 8, we also saw ideal functionalities for key-exchange and secure-channels. In fact, we can show that these natural and

[1]Note that key-exchange *can* be realized in the plain UC model, but it is arguably not two-party in the traditional sense, since two honest parties work together *against* a third dishonest party.

practical protocols are realized securely by ISO 9798-3, IKEv1, IKEv2, SSL/TLS, and other popular algorithms (provided their underlying assumptions are secure). Some natural protocols for which we do not have good intuition on their ideal functionalities are: password-based key exchange, symmetric encryption, and message authentication. What should *ideally* happen in these protocols?

3 UC Commitments

Now that we have a feeling for the UC framework and some general idea of the limitations of realizing protocols that are secure in such a real world situation, we will see a concrete impossibility result. It is impossible to realize UC-secure commitments in the plain UC framework. Let's first review the ideal functionality for commitments.

3.1 The Commitment Functionality, F_{com}

We define the stateful, commitment functionality, F_{com}, to behave as follows:

- Upon receiving $(sid, C, V,$"commit"$, x)$ from (sid, C), do:
  1. Record (sid, x).
  2. Output $(sid, C, V,$"receipt"$)$ to (sid, V).
  3. Send $(sid, C, V,$"receipt"$)$ to S.
- Upon receiving $(sid,$"open"$)$ from (sid, C), do:
  1. Output (sid, x) to (sid, V).
  2. Send (sid, x) to S.
  3. Halt.

This functionality is reactive, since F_{com} waits for two separate messages from C – one to commit and one to open – before halting. Thus, it cannot be captured as function evaluation. Note that in F_{com}, we assume that once a commitment is opened it is public information (i.e., freely known by any dishonest party).

3.2 Impossibility of Realizing F_{com} in the Plain Model (i.e., without a CRS)

In this section, we will prove that F_{com} is unrealizable in the plain UC framework. We make the following restrictions on any protocol attempting to realize F_{com}:

1. **Terminating.** A protocol is considered to terminate if, when run between two honest parties, some output is generated by at least one of the parties.

2. **Bilateral.** A protocol is bilateral if exactly two parties participate in it.

Considering the three restrictions above, we arrive at the following impossibility result:

Theorem 3.1 *There exist no terminating, bilateral protocols that securely realize F_{com} in the plain UC framework.*

This theorem holds even in the F_{auth}-hybrid model; however, we will see in the next lecture that commitments become possible to realize in the F_{crs}-hybrid model (where each party has access to a common random string).

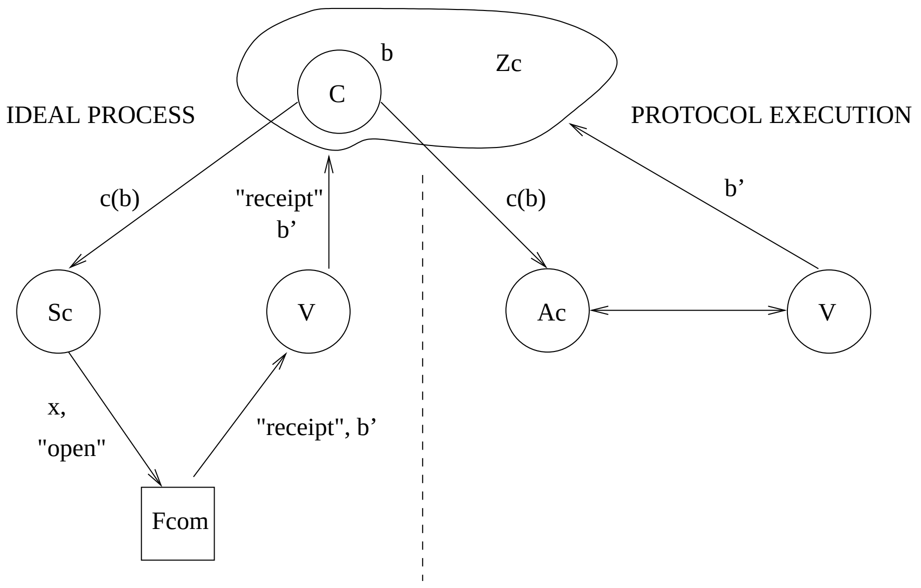

Figure 2: Pictorial view of the corrupt committer case for both the commit and open stages.

Proof. We will prove by contradiction. Suppose that P is a protocol that realizes F_{com} in the plain UC framework. This means that for all adversaries A there exists a simulator S for all environments Z such that Z cannot distinguish between an ideal run of F_{com} with S and the protocol execution of P with A. Since there must exist a simulator S for *any* adversary A, we will consider the "dummy" adversary A_d that simply passes messages between the parties.

(I) Corrupt Committer. First, let us consider the following environment Z_C and real-life adversary A_C that controls the committer C (illustrated in Figure 2):

- A_C is the dummy adversary. It reports to Z_C any message received from the verifier V, and send to V any message provided by Z_C.
- Z_C chooses a random bit b, and runs the code of the honest C by instructing A_C to deliver all the messages sent by C. Once V outputs "receipt", Z_C runs the opening protocol of C with V, and outputs 1 if the output bit b' generated by V is equal to b.

From the security of P, there exists an ideal-process adversary S_C such that $IDEAL^{Fcom}_{Sc,Zc}$ $\sim EXEC_{P,Ac,Zc}$, but this is a little weird. First, observe that in the real-life model, b' (the output of V) is almost always the same as the bit b that secretly Z chose. Consequently, in the ideal process, b and b' should almost always be the same. This means that the bit x that S provides to F_{com} at the commitment phase is almost always equal to b. If this doesn't seem odd yet, keep reading.[2]

[2]Essentially, S_C must be extracting the bit b out of the commitment Y. Since S_C has no more power than V, the verifier could simply hijack S_C's code and extract the b itself.

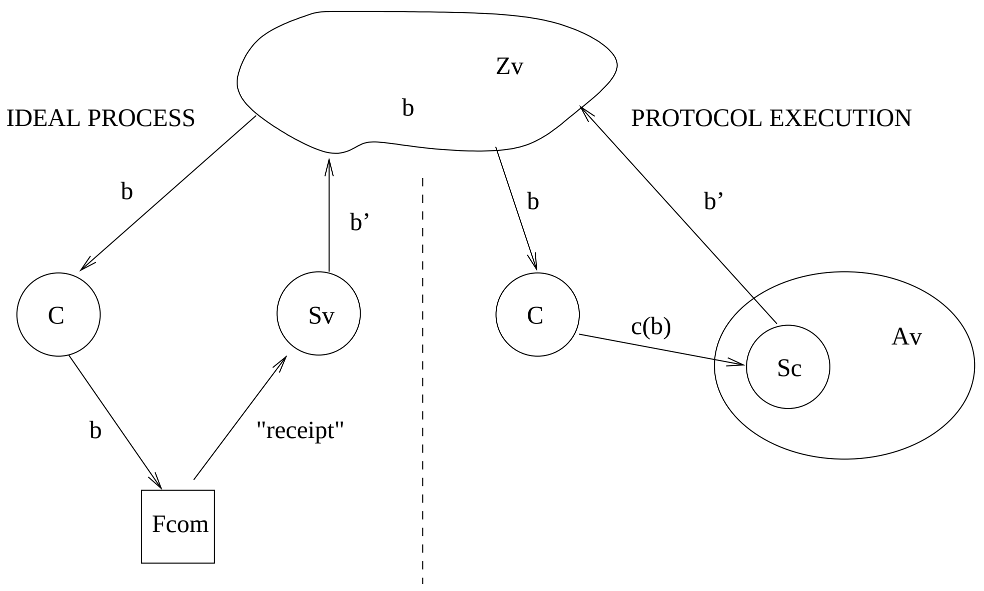

Figure 3: Pictorial view of the corrupt verifier case for only the commit stage.

(II) Corrupt Verifier. Now, with this S_C in mind, let us consider the following environment Z_V and real-life adversary A_V that controls the verifier V (illustrated in Figure 3):

- Z_V chooses a random bit b, gives b as input to the honest commiter C, and outputs 1 if the adversary A_V outputs a bit $b' = b$.
- A_V runs S_C. Any message received from C is given to S_C, and any message generated by S_C is given to C. When S_C outputs a bit x to be given to F_{com}, A_V outputs x and halts.

Notice that the view of S_C when run by A_V is identical to its view when interacting with Z_C in the ideal process for F_{com}. Consequently, from part (I), we have that in the run of Z_V and A_V it is the case that $b = b'$ almost always. However, when Z_V interacts with *any* simulator S in the ideal process for F_{com}, the view of S is independent of b. Thus, Z_V outputs 1 with probability at most one half. This contradicts the assumption that P securely realizes F_{com}.

$\square$

References

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). *STOC 1988*, pages 1–10, 1988.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *FOCS 2001*, pages 136–145, 2001.

[CDN01] Ronald Cramer, Ivan Damgard, and Jesper Buus Nielsen. Multi-party computation from threshold homomorphic encryption. *EUROCRYPT 2001*, pages 280–299, 2001.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. *CRYPTO 2001*, pages 19–40, 2001.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. *STOC 1996*, pages 639–648, 1996.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *EUROCRYPT 2004*, 2004. To appear.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. *STOC 2002*, pages 494–503, 2002.

[DCO$^+$01] Alfredo DeSantis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero-knowledge. *CRYPTO 2001*, pages 566–598, 2001.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). *STOC 1991*, pages 542–552, 1991.

[DG03] Ivan Damgard and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. *STOC 2003*, pages 426–437, 2003.

[DN02] Ivan Damgard and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. *CRYPTO 2002*, pages 581–596, 2002.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, 1988.

[HMQ04] Dennis Hofheinz and Joern Mueller-Quade. Universally composable commitments using random oracles. *Theory of Cryptography Conference*, 2004.

[RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multi-party protocols with honest majority (extended abstract). *STOC 1989*, pages 73–85, 1989.

[RS91] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. *CRYPTO 1991*, pages 433–444, 1991.