6.897: Advanced Topics in Cryptography March 18, 2004

Lecture 13,14: JUC; UC Signatures and Authentication

Lecturer: Ran Canetti *Scribed by: Steve Weis and Yoav Yerushalmi*

1 Review of Last Week and Outline for This Week

Last week we showed how to realize F_{ZK} in the F_{COM}-hybrid model. We also showed how to realize any "standard" functionality in several settings, including:

- the F_{auth}- hybrid model for semi-honest adversaries,
- the (F_{auth}, F_{crs})- model for Byzantine adversaries,
- the F_{auth} hybrid model for Byzantine adversaries with an honest majority.

This week we will look at the motivation, formulation, proof and applications of Universal Composition with Joint State (JUC). We will develop a UC formulation for signature schemes and show its equivalence with CMA-security. Finally, we will look at how to achieve authenticated communication.

One question which arose from last week's lectures was whether we really needed extremely long common reference strings (CRS) to realize F_{COM}. Indeed, a naive use of CRS would entail:

- A copy of F_{CRS} per copy of F_{COM}.
- $O(k)$ copies of F_{COM} per copy of F_{ZK}.
- $O(r)$ copies of F_{ZK} per copy of F_{CP} to complete a protocol with r rounds.
- $O(n)$ copies of F_{CP} for n parties.

One question is whether it is actually necessary to use so many copies of the CRS. We could consider a single copy of F_{MCOM} which uses a single CRS. This would not take advantage of the UC theorem, since we'd have to analyze the security of the entire protocol, including copies of F_{ZK} and F_{CP} as a single, monolithic unit. As it stands, the UC theorem is not equipped to handle protocols sharing some joint state.

Another example problem where protocols share some joint state are multiple key-exchange protocols based on the same signature scheme. That is, multiple key-exchange protocol instances using the same global parameters, such as public keys.

2 Formalization of Multi-Instance Composition

Taking a more abstract view of the problem, suppose we have:

- A protocol Q in the F-hybrid model that uses multiple copies of some functionality F. For example, F could be F_{COM} and Q the Blum ZK protocol for Hamiltonicity.

- A protocol P that realizes (in a single instance) multiple independent copies of F. For example, P could realize F_{MCOM}.

We want to know whether we can compose these two protocols and maintain security. To do so, we will now formalize a multi-instance extension of ideal functionalities as follows. Let F be an ideal functionality. Then a multi-session extension of F, denoted FF proceeds as follows:

- FF runs multiple copies of F. Each copy of F has its own session identifier $ssid$.
- FF expects all of its inputs to be of the form $(sid, ssid, \ldots)$, where sid is the session id of FF.
- An incoming message with a given $ssid$ s is routed to the copy of F whose $ssid$ is s. If no such copy of F exists, FF will invoke a new F and assign it the $ssid$ s.
- Whenever a copy of F generates some output $(ssid, msg)$, then FF will simply add its sid to that output and forward $(sid, ssid, m)$ to the intended recipient.

Consider the following two examples:

- F_{MCOM}:
  1. Upon receiving $(sid, cid, C, V, \text{“}commit\text{”}, x)$ from (sid, C) do:
     (a) Record (cid, x).
     (b) Output $(sid, cid, C, V, \text{“}receipt\text{”})$ to (sid, V).
     (c) Send $(sid, cid, C, V, \text{“}receipt\text{”})$ to S.
  2. Upon receiving $(sid, cid, \text{“}open\text{”})$ from (sid, C) do:
     (a) Output (sid, cid, x) to (sid, V).
     (b) Send (sid, cid, x) to S.

  This F_{mcom} functionality is essentially the same as FF_{com}.

- FF_{CRS} (with distribution D):
  1. Upon receiving $(sid, ssid, pid, \text{“}crs\text{”})$ from (sid, pid) do:
     (a) If there is a recorded pair $(ssid, v)$ then output v to (sid, pid) and send $(ssid, pid, v)$ to the adversary.
     (b) Else, choose a value v from D, record $(ssid, v)$ and continue as in step 1.(a).

3 Universal Composition with Joint State (JUC)

The composition operation on protocols with joint state starts with:

- A protocol Q in the F-hybrid model (that may run multiple copies of F).
- A protocol P that securely realizes FF.

Construct the composed protocol, denoted$Q^{[P]}$:

- At the first activation of $Q^{[P]}$, each party invokes a copy of P with some fixed sid, e.g. $sid = 0$.
- Whenever protocol Q calls a copy of F with input $(sid = (s, x))$, $Q^{[P]}$ calls P with input $(sid = 0, ssid = (s, x))$.
- Each output $(0, s, y)$ of P is treated as an output (s, y) coming from the copy of F with $sid = s$.

Theorem 1 (JUC: UC with Joint State) *Let Q be a protocol in the F-hybrid model, and let P be a protocol that securely realizes FF. Then protocol $Q^{[P]}$ emulates protocol Q.*

In other words, for any adversary A there exists an adversary H that for any environment Z we have

$$EXEC^{F}_{Q,H,Z} \approx EXEC_{Q^{[P]},A,Z}$$

Corollary 1 *If Q securely realizes some ideal functionality G, then so does protocol $Q^{[P]}$.*

3.1 Application of JUC

One application of the JUC theorem is in the construction of [CLOS]. In this setting, F is F_{COM} and FF is F_{MCOM}:

- Can write and realize each functionality (e.g. ZK, C&P, general compiler) as a single instance,
- Can use the UC theorem to obtain a composed protocol Q in the F_{COM}-hybrid model. Protocol Q uses many copies of F_{COM}
- Can then use the JUC theorem to compose Q with a single copy of the protocol that realizes F_{MCOM}, thus using a single copy of the CRS.

3.2 Proof of the JUC Theorem

The general proof outline will be to define a protocol Q' that is secure in the FF-hybrid model and show:

1. Protocol $Q^{[P]}$ is identical to Q'^{P}.
2. Protocol Q'^{P} emulates Q'.
3. Protocol Q' emulates Q.

The definition of Q' is fairly straightforward. Q' essentially runs Q, then prepends its own identifier to any messages sent by Q. More specifically, protocol Q' in the FF-hybrid model is identical to Q except:

- Q' uses a single copy of FF with a fixed sid s' (For instance $s' = 0$ or s' is the sid of Q' with some fixed prefix.)

- Any input x sent by Q to copy s of F is replaced with a call (s', s, m) to FF.
- Any output (s', s, y) from FF is treated as an output y coming form copy s of F.

By the UC theorem, protocol Q'^P emulates Q'. However, Q'^P is just another way of describing $Q^{[P]}$, thus $Q^{[P]}$ emulates Q'. It remains to show that Q' emulates Q.

Let A' be an adversary interacting with Q' in the FF-hybrid model. Construct an adversary A that interacts with Q in the F-hybrid model. It can be verified that $EXEC^{FF}_{Q',A',Z} = EXEC^{F}_{Q,A,Z}$ for all Z.

Adversary A runs A':

- Messages sent by A' to parties running Q' are forwarded to the actual parties running Q.
- Messages from the parties running Q are forwarded to A'.
- For each message (s', s, m) sent by A' to FF, A sends the message (s, m) to copy s of F.
- Whenever A gets a message m from a copy of F with $sid = s$, it forwards message (s', s, m) from FF to A'.
- Whenever A' corrupts a part, A corrupts the same party and reports the obtained information to A.

4 General Protocols in CRS model

Assume we had a protocol that realizes FF_{CRS} in the F_{CRS}-hybrid model, using only a single instance of F_{CRS}. Then it would suffice to construct protocols where each instance uses its own copy of F_{CRS}. For instance, realizing F_{COM} would be sufficient; we wouldn't need F_{MCOM}.

Several results from [CR03] are relevant to this setting:

- Any protocol that realizes FF_{CRS} in the F_{CRS}-hybrid model using only a single copy of F_{CRS} must be interactive, i.e. each party should send at least one message for each generation of a new common string.
- Using the Blum 3-move coin-tossing protocol, we can realize FF_{CRS} in the F_{MCOM}-hybrid model using only a single copy of F_{CRS}.
- Using protocol UCC, we get the desired results. (But could not get rid of the protocol UCC.)

5 Applications of JUC to Signature-Based Protocols

Another case where multiple protocol instances use the same subroutine are cases of protocols based on signature schemes. For example, signature-based message authentication,

key-exchange, or Byzantine Agreement. In all of these cases, protocols use long-term signature keys for multiple protocol sessions. That is, signature keys can be considered joint state shared among protocols.

Our goal is to define and analyze signature-based protocols for a single session (i.e. single session-key) and then use JUC to deduce that the multi-session interaction (using a single long-term signature module) is secure. to do that, we first need to be able to formalize the signature mechanism as an ideal functionality.

5.1 Digital Signatures as Ideal Functionalities

Digital signatures are typically thought of as a tool within protocols, rather than a "protocol" by itself. Still, it is useful and instructive to treat digital signatures as a protocol with a specified ideal functionality. Potential benefits include modularity of analysis, reasserting the adequacy of existing security notions, and providing a bridge to formal analysis of protocols.

The question is how exactly to formalize digital signatures. There are two main approaches. One is to define signatures as a stand-alone primitive, as in [Can01, CK02, CR03, BH03, Can03]. The other is to define signatures as a part of a more complex functionality that also provides other services, as in [BPW03]. We'll focus on the stand-alone approach because it is more modular.

5.2 Attempt 1

The following is our first attempt at defining an ideal functionality for a digital signature scheme:

1. On input $(sid, \text{“}Keygen\text{''})$ from party (sid, S) , register party (sid, S) as the signer.
2. On input $(sid, \text{“}sign\text{''}, m)$ from (sid, S) record m.
3. On input $(sid, \text{“}verify\text{''}, m)$ from any party, return $(sid, \text{“}yes\text{''}/\text{''}no\text{''})$ according to whether m has been recorded.

This definition is too idealized. Any protocol that realizes this functionality will have to take care of transmitting signatures and public keys between parties. This would not meet our intuition of a "signature scheme".

5.3 Attempt 2

The following protocol attempts to resolve the problem by specifically creating a signature key and a signature string:

1. On input $(sid, \text{“}Keygen''$) from party (sid, S) , register party (sid, S) as the signer, AND return to party (sid, S) a public key v selected at random.
2. On input $(sid, \text{“}sign'', m)$ from (sid, S) return a random signature s and record (m, s, v).
3. On input $(sid, \text{“}verify'', m, s, v')$ from any party, return $(sid, \text{“}yes''/''no'')$ according to whether (m, s, v') has been recorded.

It looks as though the functionality is no longer too ideal. In particular since the signature string and verification key are part of the interface, communicating the values from one party to another becomes the job of the calling protocol, rather than the job of the signature scheme. But this formulations still has several problems. For one thing, the public key in the ideal functionality is a random number. Many real-world systems do not do this (and we consider them secure). The scheme is also very deterministic. Specifically, the ONLY accepted signature is the one that was generated by (sid, S) as the triplet (m, s, v). For some applications, we want digital signature schemes that have multiple valid signatures for a given message. Finally, the ideal scheme is such that even an invalid signer can't pick a 'weak' signing key.

So in all, we have now made the scheme TOO restrictive. On to our next attempt.

5.4 Attempt 3

We will once again attempt to resolve the previous problems:

1. On input $(sid, \textbf{“Keygen”})$ from party (sid, S) , register party (sid, S) as the signer, forward (sid, S) to A (the simulator/adversary), obtain a public key v from A and output v to (sid, S).
2. On input $(sid, \textbf{“sign”}, m)$ from (sid, S), forward (sid, m) to A, obtain a "signature" s from A, output s to (sid, S), and record (m, s, v).
3. On input $(sid, \textbf{“verify”}, m, s, v')$ from any party, return (sid, f) where:
   - If (m, s, v') is recorded then $f = 1$.
   - If S is uncorrupted and (m, s^*, v') is unrecorded for any s^*, then $f = 0$.
   - Else forward (m, s, v') to A and obtain f from A.

First note that the second assignment of f (where S is uncorrupted) ensures that phony signatures can't be generated by A, but allows for more than one valid signature for a signed message.

It almost seems as if we've reached a good definition, except that there is still one glaring problem. Now that we have allowed A the power to decide whether a signature is valid or

not when the signer is corrupt, we have also allowed A to change his mind (for the exact same signature, he can say "yes" once and "no" at another point). This is not a desirable feature for a signature scheme.

5.5 Attempt 4

We will make a minor change to the functionality, forcing the system to stick to its decision once it is made:

1. On input $(sid, \textbf{"Keygen"})$ from party (sid, S) , register party (sid, S) as the signer, forward (sid, S) to A (the simulator/adversary), obtain a public key v from A and output v to (sid, S).
2. On input $(sid, \textbf{"sign"}, m)$ from (sid, S), forward (sid, m) to A, obtain a "signature" s from A, output s to (sid, S), and record $(m, s, v, 1)$ unless a prior record $(m, s, v, 0)$ exists, in which case output an error message.
3. On input $(sid, \textbf{"verify"}, m, s, v')$ from any party, return (sid, f) where:
   - If (m, s, v', b) is recorded then $f = b$.
   - If S is uncorrupted and $(m, s^*, v', 1)$ is unrecorded for any s^*, then $f = 0$.
   - Else forward (m, s, v') to A and obtain f from A. Record (m, s, v', f)

Sadly, although we're almost there, there's still one more minor change we need to make. The problem with the above protocol is that it doesn't actually predefine a fixed signer. In a signature scheme, the signer of a document is a known value, but in the above, other than claiming to be the role of the signer, nothing forces it. It isn't clear that only one signer is allowed.

5.6 F_{sig}

We will fix this problem by encoding who the signer is within the session ID. That way, for every signature session, the 'signer' is well-known to all parties, even if he hasn't signed anything yet. This will yield our ideal signature functionality:

F_{sig}

1. On input (sid, **"Keygen"**) from party (sid, S) , verify $sid = (sid', S)$ otherwise, ignore input. Forward (sid, S) to A, obtain a public key v from A and output v to (sid, S). This stage may only be done once.
2. On input (sid, **"sign"**, m) from (sid, S), where $sid = (S, sid')$ forward (sid, m) to A, obtain a "signature" s from A, output s to (sid, S), and record $(m, s, v, 1)$ unless a prior record $(m, s, v, 0)$ exists (in which case output an error message).
3. On input (sid, **"verify"**, m, s, v') from any party, return (sid, f) where:
   - If (m, s, v', b) is recorded then $f = b$.
   - If S is uncorrupted and $(m, s^*, v', 1)$ is unrecorded for any s^*, then $f = 0$.
   - Else forward (m, s, v') to A and obtain f from A. Record (m, s, v', f)

And voila, we have a working ideal functionality. This is the functionality we will attempt to achieve. However, it should be noted that this functionality relays all messages and signatures to the adversary, which means whatever protocol achieves it is not a secret or anonymous signature scheme.

6 Realizing F_{sig}

It turns out that it is fairly easy to make a real-world protocol that can securely realize F_{sig}. Given any signature scheme $H = (GEN, SIG, VER)$, we construct the following protocol P_H:

- When invoked with (sid, **"Keygen"**) and $pid = S$ ensure that $sid = (S, sid')$. Now run $(p, v) \leftarrow GEN(k)$. Return v to the caller (and keep p private).
- When invoked with (sid, **"sign"**, m) compute $s \leftarrow SIG(p, m)$ and return s. We allow SIG to maintain state between activations.
- When invoked with (sid, **"Verify"**, m, s, v') return $VER(m, s, v')$

We show that this scheme is indeed secure:

Theorem 2 *A scheme H is existentially unforgeable against chosen message attacks if and only if the protocol P_H securely realizes F_{sig}.*

Recall the definition of security we will use:

Definition 1 *a scheme $H = (Gen, Sig, Ver)$ is existentially unforgeable against chosen message attacks (is EU-CMA) if it has the following three properties:*

Completeness *: For all adversaries F:*

$$Pr_{(p,v)\leftarrow Gen();m\leftarrow F(v)}[Ver(m, Sig(p, m), v) = 1] \sim 1$$

Consistency *: For all* (m, s, v)*:*

$$Variance[Ver(m, s, v)] \sim 0$$

Unforgeability *: Over all* $(p, v) \leftarrow Gen(k)$*,* $(m^*, s^*) \leftarrow F^{SIG(p,*)}(v)$ *where* m^* *is not one that* F *has asked for a signature of:*

$$Pr_{above}[Ver(m^*, s^*, v) = 1] \sim 0$$

To clarify the above (it is intended to be an interactive system), we allow the adversary to know the public key, and make polynomially many queries to a signing machine. After receiving many signatures, he attempts to generate a valid new signature for a new message. Our scheme is unforgeable if his chances of doing well are negligible.

6.1 Proof that (P_H realizes F_{sig}) $\longleftrightarrow$ (H is EU-CMA secure).

- P_H **realizes** $F_{sig} \longrightarrow H$ **is EU-CMA secure**.
  For this to be true means we can show that H has the three properties of EU-CMA. We will therefore go over each property and prove it.

  Completeness : To prove this, we will set up a contradiction. Assume that H is not complete. That means that there exists some adversary for H (we call it F) that can find a message m whose signature doesn't verify correctly. So from that, we can build an environment Z and an adversary A where the environment instructs the signer to sign m and then verify the signature. In the ideal model, that message still verifies correctly (verification is independent of H). In the hybrid model, it will not, and so the environment can distinguish. This leads us to conclude that P_H doesn't securely realize F_{sig} which is a contradiction.

  Consistency : To prove, we will use the same trick. Assume that H is not consistent, so the result of the verification function varies. By a similar trick to the above, we construct an environment that calls for several verifications in a row of the same message and signature. In the ideal, we always get the same result, while in the real, we will flip-flop. The environment can distinguish, which violates the assumption of secure realization of F_{sig}.

  Unforgeability : Assume there exists a forger G for H. We will once again show how the environment can distinguish the ideal from the real (and form a contradiction). As normal behavior, the environment invokes a signer/verifier/adversary triplet. Additionally, the environment runs the forger G internally. To get a key to give to G, the environment tells the signer to run keygen (yielding only a public key v which the environment feeds to G). G now will ask for many messages m_i to be signed. For each such message, the environment will pass it to the uncorrupted signer to sign. It will get a signature s_i for each m_i, which it will hand over to G. Finally, G will generate a pair m^*, s^* which it claims to be a valid signature (and is indeed, since we claim G is a forger for H). The environment will now ask the verifier to verify the signature s^*, and output the bit that the verifier does. Note that in the ideal environment, the final signature

is NOT a valid signature since it was never recorded as such, so it will always output 0. On the other hand, in the real-world environment, the final signature IS a valid signature with non-negligible probability, so the environment can distinguish (and we contradict).

- H **is EU-CMA secure** $\longrightarrow$ P_H **realizes** F_{sig} .
  We will again do this through contradiction. Assume P_H does not securely realize F_{sig}. That means that for any simulator S there is an environment (Z) that can distinguish between an interaction with the ideal functionality F_{sig} and S, and the real protocol P_H and the dummy adversary. Specifically, we will look at the environment that can distinguish with respect to the following generic simulator S:

  - When asked by F_{sig} to generate a key, S runs $(p, v) \leftarrow Gen(k)$ and returns v (just as per the protocol).
  - When asked by F_{sig} to generate a signature s on a message m, S runs $s \leftarrow Sig(p, m)$ and returns s.
  - When asked by F_{sig} to verify a signature (m, s, v'), S runs $f \leftarrow Ver(m, s, v')$ and returns f.

  As can be seen, S follows the exact decision behavior of the original protocol P_H.

  Let $\mathbb{B}$ be the event that in a run of Z and S in the ideal model, the signer never signed m, and yet still an $(sid,$ **"Verify"**$, m, s, v)$ activation answered with a 1. It can be verified that if event $\mathbb{B}$ doesn't occur, Z's view of the ideal and real executions are statistically close (these can be randomized signatures). We know that Z can distinguish, so $\mathbb{B}$ must occur with non-negligible probability. Using this knowledge, we can now construct a forger G for H (thereby contradicting our assumption that H is EU-CMA secure).

  Here is how G works. It first takes the code for Z and runs it internally:

  - When Z attempts to start a signer with "KeyGen", G doesn't start a signer, but instead feeds Z the output for the signer, that being the value v that G is trying to break.
  - When Z attempts to get the signer to sign a message m, G intercepts this and passes the m to its signing oracle. It will get a valid signature s, which it will hand back to Z (looking like the valid signature coming back from the signer).
  - When Z attempts to verify a signature (m, s, v), G can check whether (m, s, v) is a forgery (this is event $\mathbb{B}$). If it is, then it outputs (m, s, v). Otherwise, it continues to run Z.
  - If/When Z asks to corrupt the signer, G aborts.

  We know that $\mathbb{B}$ occurs with non-negligible probability, so G generates a valid forgery with non-negligible probability, which is a contradiction on the assumption that H is EU-CMA secure.

Corollary 2 *From the above proof we see that P_H is adaptively secure iff it is non-adaptively secure*

Clearly if it is adaptively secure, it is non-adaptively secure (special case). To see the other direction, simply note G aborts on a "corruption" message from Z irrelevant of when it happens, and yet our contradiction still holds.

7 Authenticated Communication using F_{sig}

We wish to establish an authenticated communication network on top of an unauthenticated network. Using this, we can provide F_{auth} as an underlying functionality. We will follow this plan:

- We define a registry functionality F_{reg} which allows parties to "register" their public keys for others to securely retrieve.
- We realize F_{auth} in the $(F_{reg}, F_{sig}) - hybrid$ model by:
  - Creating a certification functionality F_{cert} that provides a binding between signatures and parties.
  - Realize F_{cert} in the $(F_{reg}, F_{sig}) - hybrid$ model.
  - Realize F_{auth} in the $F_{cert} - hybrid$ model.
- We will then authenticate multiple messages using a single key pair by:
  - Defining FF_{cert}
  - Realize FF_{cert} using many calls to F_{cert}.
  - Use the JUC theorem to combine.

7.1 The "Public Registry" Functionality F_{reg}

This functionality allows a party to register a value (nominally its public key) to a system in a way that allows any party to ask for that value:

F_{reg}

1. When receiving $(sid, \textbf{"register"}, v)$ from party (sid, S), verify that $sid = (S, sid')$, send (sid, S, v) to the adversary, and record (S, v).
2. When receiving $(sid, \textbf{"retrieve"}, S)$ from any party, return (sid, S, v) if there is a record (S, v) else return $(sid, S, -)$.

Note that there is no verification of the public key being anything (no proof of knowledge of p or the like). Additionally, the way our definition works, the registrant is part of the sid, so only one person can register their key per functionality.

7.2 The Certification Functionality F_{cert}

F_{cert}

1. On input $(sid, \textbf{“sign”}, m)$ from (sid, S) where $sid = (S, sid')$, forward (sid, m) to A, obtain a “signature” s from A, output s to (sid, S) and record $(m, s, 1)$. Verify that no $(m, s, 0)$ record exists.
2. On input $(sid, \textbf{“verify”}, m, s)$ from any party, return (sid, f) where:
   - If (m, s, b) is recorded, then $f = b$.
   - If S is uncorrupted and $(m, s^*, 1)$ is unrecorded for any s^*, then $f = 0$.
   - Else forward (m, s) to A and obtain f from A, recording (m, s, f)

Note that F_{cert} is similar to F_{sig} except there is no more key generation, and instead it is all done with respect to the signers identity (which is encoded in the sid).

7.3 Realizing F_{cert} in the $(F_{reg}, f_{sig}) - hybrid$ model

The following is the protocol that realizes the F_{cert} functionality:

1. At first activation, signer (sid, S) ensures that $sid = (S, sid')$ and calls F_{sig} with $(sid.0, \textbf{“keygen”})$, obtains v, and calls F_{reg} with $(sid.1, \textbf{“register”}, v)$.
2. When activated with input $(sid, \textbf{“sign”}, m)$, (sid, S) verifies $sid = (S, sid')$ and calls F_{sig} with $(sid.0, \textbf{“sign”}, m)$, obtaining a signature s which it outputs.
3. When activated with input $(sid, \textbf{“verify”}, m, s)$ where $sid = (S, sid')$, the activated party calls F_{reg} with $(sid.1, \textbf{“retrieve”}, S)$ obtains the public key v, and calls F_{sig} with $(sid.0, \textbf{“verify”}, m, s, v)$. It outputs the result of that call.

The simulation is perfect for the above (all the smarts/computation are done in the underlying protocols). Similarly, the security is unconditional.

7.4 Realizing F_{auth} in the $F_{cert} - hybrid$ model

The following is the authenticated message transmission functionality F_{auth}:

F_{auth}

1. Receive input (sid, S, R, m) from party (sid, S).
2. Output (sid, S, R, m) to party (sid, R)
3. Send (sid, S, R, m) to the adversary/simulator.
4. halt

And the protocol that achieves it:

1. When activated with input (sid, S, R, m), party (sid, S) calls F_{cert} with $(S.sid, \textbf{“sign”}, m.R)$, obtains a signature s, and sends (sid, S, m, s) to (sid, R).

2. When receiving message (sid, S, m, s), (sid, R) calls F_{cert} with $(S.sid, \textbf{“verify”}, m.R, s)$. If the result is 1, then output (sid, S, R, m).

Again the simulation is perfect, and the security is unconditional. Also, although most real protocol use nonces as part of the message to ensure unique messages, this system uses *sid* values, which are supposed to be unique (indeed, a natural way to obtain unique sid's is to exchange nonces and concatenate the nonces into the sid).

Next lecture we will discuss how to achieve authenticate multiple messages using a single verification key (by using the JUC theorem).

References

[BH03] Michael Backes and Dennis Hofheinz. How to Break and Repair a Universally Composable Signature Functionality. 2003.

[BPW03] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Universally Composable Cryptographic Library. 2003.

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

[Can03] Ran Canetti. Bib file pending. 2003.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT 2002*, pages 337–351, 2002.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Advances in Cryptology - CRYPTO 2003*, pages 265–281, 2003.