6.897: Advanced Topics in Cryptography Feb 5, 2004

Lecture 1,2: Universal Composability

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

1 Handouts

There were two handouts: handout #1 was General Information, and handout #2 was the planned lecture schedule until spring break.

2 Administrivia

The first half of the course will be taught by Ran Canetti, and will cover topics related to Security-Preserving Composability (see handout 2 for tentative schedule of lectures). The second half, starting after spring break, will be taught by Ron Rivest, and will cover varying topics including voting protocols, extractors, public-key infrastructure, and digital signatures.

There are some very good slides of the lecture which cover much the same material as is in these notes.

3 Overview

In the first half of the course, we will be covering the basic foundations of cryptographic protocols. Our goals are to provide some theoretical foundations of what it means to be a secure cryptographic protocol, including what it means to be secure in general. We will see how to preserve security when protocols are combined, and we will see some basic constructions of protocols whose security is preserved through composition.

The class will have an overall definitional and foundational slant, and will be less focused on in-depth proofs. We would like to be able to define what it is we want from a protocol, and figuring out what it is we need to prove about them. It will stress much more the conceptual points and considerations, and will not be very concerned with technical details.

4 Definition of Security

An important question to ask when trying to define security for a given task is "What do we want from our definition"?

- It should be mathematically rigorous. It should be well-defined how the system works and is modeled, so that we can then determine how to model a protocol, and if a given modeling of a protocol fits within our definition or does not.
- It should provide a convenient abstraction (**primitive**) that matches our intuition for the task.
- Our definition should capture "all realistic attacks" in the expected execution environment.
  - What are our network characteristics? (synchronous/asynchronous, reliable, etc.).
  - What power does the attacker have? (can he control protocol participants? can he control the network links? can he inject data or merely listen? etc.)
  - What are the possible inputs to the system, including exception conditions.
  - What other protocols are (or might be) running in the same system, and how can the protocols interact.
- It should guarantee security even if the primitive is used elsewhere (for example, as a subroutine of another protocol). That is, it should maintain security under composition.
- It should not be overly restrictive.
- We need to base it on the functionality of the candidate protocol, and not on the structure of the protocol.
- It would also be nice if our framework was able to support multiple tasks, and was both conceptually and technically simple.

It's clear that the security requirement and composability are intertwined, leading us to include a notion of "secure composability" in the basic desiderata.

5 First Candidate : Multiparty Secure Function Evaluation

To demonstrate these points we investigate the "classic" task of secure multiparty function evaluation. The basic concept is that we have multiple people, each with their own private value, who are trying to compute some function on those values, without leaking anything other than the result of the function. More specifically, we have:

- n parties $(p_1, p_2, \ldots, p_n)$ with $n > 1$, where each p_i has an input $x_i \in \mathbb{D}$. Some parties may be corrupted. We will assume that the corruptions happen in the beginning (static corruptions).
- We have a probabilistic function $f : \mathbb{D}^n \times \mathbb{R} \Rightarrow \mathbb{D}^n$
- We have a communications network that allows participants to send messages to each other.

We want a secure protocol where at the end, each participant p_i has a value $f(x_1, \ldots, x_n, r)_i$. We want to ensure:

Correctness : All honest participants get a correct evaluation of the function.

Secrecy : All corrupted parties learn nothing more than what they knew before, plus their prescribed outputs.

Here are some example functions to illustrate:

- $F(x_1, \ldots, x_n) = x_1 + \ldots + x_n$
  a simple sum function. (All parties get the same value)
- $F(x_1, \ldots, x_n) = MAX(x_1, \ldots, x_n)$
  a max-value function.
- $F(-, \ldots, -) = r \Leftarrow_U \mathbb{D}$
  a simple coin toss function. Here the main requirement is that the output remains unbiased in spite of malicious behavior.
- $F((x_0, x_1), b) = (-, (x_b; b \in_R 0, 1))$
  This is 1-of-2 oblivious transfer. Party 2 learns one of two values that party 1 had, and party 1 doesn't know which value party 2 learned.
- $F_R((x, w), -) = (-, (x, R(x, w)))$ where $R(x, w)$ is a binary relation.
  This is a modelling of Zero-Knowledge (we have correctness and soundness). This is also a proof of knowledge of a witness.

6 Formalizing

We would like to be able to formalize our requirements for correctness and secrecy.

6.1 Correctness

We have a difficulty in deciding how the function should be compute in the case of the corrupt parties.

- if we require that f is computed on input values fixed at the start, then we have an unrealizable definition. A corrupt party might simply choose to replace its input with another value, and there is no way for the party computation to account for its pre-set value.
- If we allow the corrupt party to "choose" its input, then we have a problem. Specifically, imagine a protocol for computing the sum of all inputs. Lets use two parties: Party one sends its secret input to party two, who replies with the sum of both. While this protocol is both "correct" and "secret", we would not call this protocol secure, since party 2 has full control of the result.

We need an "input independence" property, which unites secrecy and correctness.

6.2 Secrecy

Similarly, we would like to define secrecy. One possible attempt is the following: "It should be possible to generate the view of the corrupted parties given only their inputs and prescribed outputs".

However, the following counter-example demonstrates a flaw:

- Function: $F(-,-) = (r \Leftarrow_U \mathbb{D}, -)$. That is party 1 gets a random bit, and party two gets nothing.
- Protocol: P_1 chooses $r \Leftarrow_U \mathbb{D}$ and sends r to P_2.

As is pretty obvious, this protocol is not secret (since P_2 learns r). Yet, it is possible to generate P_2's view (it is just a random bit).

We need to consider the outputs of the corrupted parties together with the outputs of the uncorrupted parties. This forces correctness and secrecy to be combined.

7 General Definitional Approach

We will use the following definitional paradigm of a secure protocol [GMW87]

Posit 1 *A protocol is secure for some task if it "emulates" an "ideal setting" where participants hand their inputs to a "trusted party" who locally computes the desired outputs and hands them back to the participants.*

Several formalizations of this idea exists (Goldwasser-Levin90 [GL90], Micali-Rogaway91 [MR91], Beaver91 [B91], Pfitzmann-Waidner94 [PW94], Canetti00 [Canetti00], Dodis-Micali00 [DM00], ...). We will use the definition of Ran Canetti's [Canetti00]. Plan:

- Describe the model for protocol execution ("real-life model")
- Describe the "ideal process" using a trusted party
- Describe a notion of "emulating" the ideal process without the trusted party.

In class, we will discuss the definition in the case of a synchronous network with both active (byzantine) and static (non-adaptive) adversaries. We will rely on computational security (requiring that our adversary and the "distinguisher" are poly-time limited). Our network will provide authenticated, but not secret, communication.

8 Definitions, Notations, and other Preliminaries

Definition 1 *A* **distribution ensemble** $D = \{D_{k,a}\} \;\; ; \;\; k \in \mathbb{N}, a \in \{0,1\}^*$ *is a sequence of distributions, one for each value of* k, a. *We will only consider binary ensembles (where each* $D_{k,a}$ *is over* $\{0,1\}$*).*

Once we have ensembles, we can define relationships between ensembles:

Equality : $D = D' \iff \forall k, a \;:\; D_{k,a} = D'_{k,a}$

Statistical Closeness : $D \sim D' \iff \forall c, d > 0\ \exists k_0\ s.t.\ \forall k > k_0, \forall a\ \ |a| < k^d$:

$$\left[\mathsf{Pr}_{x \Leftarrow D_{k,a}}[x = 1] - \mathsf{Pr}_{x \Leftarrow D'_{k,a}}[x = 1]\right] < k^{-c}$$

Now we define a multiparty function in a way that allows the adversary to provide it own input and receive its own output:

Definition 2 *An n-**party function** is a function $f : \mathbb{N} \times \mathbb{R} \times (\{0,1\}^*)^{n+1} \Rightarrow (\{0,1\}^*)^{n+1}$*

Definition 3 *An **interactive turing machine (ITM)** is a turing machine with special-purpose tapes:*

- *The standard computation tape*
- *The incoming communication tape. Data is written on this tape by other ITMs in an append-only fashion. It is known what ITM wrote the data on the tape.*
- *The incoming subroutine output tape is a tape used for receiving inputs from subroutines (other ITMs).*
- *The identity tape. This holds a static value that the ITM can use to identify itself (each ITM has a different value).*
- *The security parameter tape. A read-only tape containing the security parameter.*
- *A random tape containing random bits.*
- *The output tape which is where the ITM writes its final result in the computation before halting.*

An ITM is "activated" whereupon it runs some computations, reads and writes to/from tapes, and then goes into either a waiting or a halted state. We will consider poly-time ITMs, which are ITMs where the overall number of steps taken is polynomial in the security parameter plus the overall input tape length.

Definition 4 *a **system** of interacting ITMs is a set of ITMs (fixed number), plus a set of write-permission rules (which ITM can write to what other ITMs' communication tapes). a **run** of a system $(M_0, \ldots, M_m)$ is:*

- *An initial ITM M_0 which starts with some external input*
- *In each activation an ITM may write to the communication tapes of other ITMs IF the permissions allow it.*
- *All ITMs whose tapes are written to during an activation are appended to an activation queue. When the currently activated ITM enters a waiting state, the next ITM on the queue is activated.*
- *The output of the system is the final output of the initial ITM M_0.*

9 "Real-Life Model" for Protocol Execution

We define a "real-life" model for protocol execution that will help encompass our security and correctness requirements. This model has the following participants:

- An n-party protocol $P = (P_1, \ldots, P_n)$, for $n > 1$, where each P_i is an ITM.
- An adversary A, controlling a set of "bad parties" $B \subset P$. The adversary gets to run arbitrary code on the "bad parties".
- The environment Z. The environment is modeled as an initial ITM which starts with some external input.

The parties, P, A and Z interact by the following computational process:

- The environment Z receives external input z.
- Z gives input a to the adversary A and the input x_i to each P_i.
- While any party P is active:
  - Good parties, i.e. $P - B$, generate messages for the current round.
  - The adversary A monitors all messages and generates messages for the bad parties B.
  - The adversary A delivers all messages generated by the good parties.
- Prior to halting, all parties P and A send their outputs to Z.
- Z generates an output bit $b \in \{0, 1\}$.

There are a couple caveats with this model. First is that messages are authenticated, so the adversary cannot spoof legitimate good party messages. The adversary must also deliver messages in their proper rounds. It is not allowed to deliver future messages before an old round completes. Finally, "honest, but curious" parties are still considered to be "bad".

We introduce some notation for this model:

- $EXEC_{P,A,Z}(k, r, z)$: Output of Z on input z, randomness r and security parameter k after interacting with A and P.
- $EXEC_{P,A,Z}(k, z)$: Output of Z on input z and security parameter k after interacting with A and P over uniform randomness.
- $EXEC_{P,A,Z}$: The ensemble of distributions $\{EXEC_{P,A,Z}(k, z)\}$ with $k \in \mathbb{N}$ and $z \in \{0, 1\}^*$.

10 The Ideal Process for Evaluation of f

We define another system of interacting ITMs to help model the computation of an arbitrary function f:

- A set of n "dummy parties", $P = (P_1, \ldots, P_n)$, for $n > 1$.
- An adversary S, controlling a set of "bad parties" $B \subset P$.
- The environment Z.
- A "trusted third party" F for evaluating f.

These parties interact by the following computational process:

- Z receives external input z.
- Z gives input a to the adversary S and the input x_i to each good party P_i.
- Good parties, $P - B$, hand their inputs to F.
- Bad parties, B, send whatever S tells them to F. S also sends its own input to F.
- F evaluates f and distributes the output to P and S.
- Good parties output the value they received from F and bad parties output whatever S tells them to output.
- S and P write their outputs to Z's output tape.
- Z generates an output bit $b \in \{0, 1\}$.

We use the following notation for this model:

- $IDEAL^f_{S,Z}(k, r, z)$: Output of Z on input z, randomness r and security parameter k after interacting with F and S.
- $IDEAL^f_{S,Z}(k, z)$: Output of Z on input z and security parameter k after interacting with F and S over uniform randomness.
- $IDEAL^f_{S,Z}$: The distribution ensemble $\{IDEAL^f_{S,Z}(k, z)\}$ with $k \in \mathbb{N}$ and $z \in \{0, 1\}^*$.

11 B-Secure Definition, Implications and Variants

Let B be a collection of subsets of $\{1, n\}$. An adversary is B-limited if the set B of parties it corrupts is in B.

Definition 5 *Protocol P B-emulates the ideal process for f if for any B-limited adversary A there exists an adversary S such that for all Z we have:*

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

We say that a protocol "B-securely realizes f" if it meets the above definition. This definition essentially means that Z cannot tell with more than a negligible probability whether it is interacting with A and real parties running P, or with S and the ideal process for f. Another way to say this is that whatever damage A can do to the parties running the protocol can also be done in the ideal process.

This definition implies: (1) Correctness - for all inputs, the good parties output the "correct function value, (2) Secrecy - whatever A computes can be computed given only the prescribed outputs, and (3) Input independence - Bad party inputs are chosen independently of the inputs of the good parties.

One question is what happens if you have an adversary that does not send its output to the environment's output tape. This isn't an issue because the definition is quantified over all adversaries.

There are several other equivalent formulations. For example, Z can output an arbitrary string (rather than one bit) and two executions will still be indistinguishable. Z and A can also be deterministic or we can change the order of quantifiers; S can depend on Z rather than vice versa.

There are many possible variants of these models:

- Passive Adversaries: Semi-honest corrupted parties continue to run the original protocol.
- Secure channels vs. Unauthenticated channels
- Unconditional Security: Allow Z and A to be computationally unbounded. (Except S need to remain polynomial in Z, A, P.)
- Perfect Security: Z's outputs in the two runs should be identically distributed.
- Adaptive Security: Both A and S can corrupt parties as the computation proceeds. Z learns about corruptions.

12 Protocol Composition

So far we have modeled only "stand-alone" security. This entails only a single execution of a single protocol on a network with no other parties or network activity. Clearly, this model does not account for many of the complexities of the real world. We consider what happens when secure protocols run in conjunction with other protocol executions.

There are several forms of running "in conjunction". It may be other executions of the same protocol, executions of other protocols, "intended" or coordinated executions, and finally "unintended" uncoordinated executions.

We must ask whether security is maintained in composition, particularly the following examples:

- Running the same protocol with same/different inputs, with same/different parties, in a serial, parallel or concurrent fashion.

- "Subroutine Composition" (or Modular Composition), where a protocol Q calls a protocol P as a subroutine. These may either be non-concurrent or concurrent.
- General Composition, which allows running arbitrary protocols in the same system without coordination.

13 Modular Composition

The model of composition we will examine is modular composition. Here we will formalize the non-concurrent case. (Concurrent modular composition will be treated later in the course.) The general idea is to model a call to a common, trusted subroutine f during the execution of a protocol Q as the execution of another protocol P. To help model this interaction, we will define a "f-Hybrid Model" protocol:

- Start with a "real-life" model protocol.
- Allow parties to have access to a trusted party F which evaluates f:
  - At pre-defined rounds, the protocol instructs all parties to send their values to F.
  - F evaluates f on the given inputs and returns outputs to the parties.
  - Once the outputs are obtained, the parties continue as usual.
- Notation: $EXEC^{f}_{P,H,Z}$ is the ensemble describing the output of Z after interacting with the protocol P and adversary H in the f-Hybrid model.
- Notes: In the "ideal call rounds", no other computation takes place. We can also model each "ideal call round" as evaluating a different function, although this doesn't add any real power.

We will now look at the composition operation, which originated in [MR91]. First we start with a protocol Q in the f-Hybrid model and a protocol P that securely realizes f. We construct the composition protocol Q^P such that each call to f is replaced by an invocation of P whose output is treated as the output of f.

There are a couple of caveats to this definition. First, only one protocol is active at a time. When P is running, Q is suspended. It is also important that all parties terminate P in the same round. Finally, if P is a protocol in the "real-life" model, so is Q^P. If P is a protocol in the f'-hybrid model for some f', then so is Q^P.

Theorem 1 *Protocol Q^P "emulates" protocol Q. That is, for any B-limited adversary A, there is a B-limited adversary H such that for any any Z we have:*

$$EXEC^{f}_{Q,H,Z} \sim EXEC_{Q^P,A,Z}$$

Corollary 1 *If protocol Q t-securely realizes function f'' in the f-hybrid model, then protocol Q^P t-securely realized f'' in the real-life model.*

The proof of this theorem will be offered in the next lecture.m

References

[B91] Donald Beaver Foundations of Secure Interactive Computing Proc. of Advances in Cryptology (CRYPTO91) 1991.

[Canetti00] R. Canetti Security and Composition of Multiparty Cryptographic Protocols. Journal of Cryptology Vol. 13, No. 1 2000.

[DM00] Y. Dodis and S. Micali Parallel Reducibility for Information-Theoretically Secure Computation Proc. of Advances in Cryptology (CRYPTO00) 2000.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson How to Play Any Mental Game. Proceedings of 19th ACM Symposium on the Theory of Computing (STOC), 1986.

[GL90] S. Goldwasser and L. Levin Fair Computation of General Functions in Presence of Immoral Majority Proc. of Advances in Cryptology (CRYPTO90) 1990.

[MR91] S. Micali and P. Rogaway Secure Computation Proceedings of Advances in Cryptology: CRYPTO'91. 1991.

[PW94] B. Pfitzmann and M. Waidner A General Framework for Formal Notions of "Secure" Systems Hildesheimer Informatik-Berichte April 1994.