0368.4162: Introduction to Cryptography Ran Canetti

Lecture 13

26 January 2009 Fall 2008

Scribes: M. Radashkovich, O. Singer

Topics for Today:

- General secure two-party and multi-party computation
- Brief overview of next semester

1 Overview

Informal Theorem: Let $m \in N$ and let $f : N \times D^m \to D^m$ be a possibly randomized function. Then there exists a protocol for m participants such that:

1. The protocol allows the parties to evaluate the function: When each participant P_i has input $n \in N$ and $x_i \in D$, at the end of the protocol each participant has output $y_i = f(n, x_1..., x_n)_i$
2. Each participant learns nothing but y_i

Furthermore, the above holds even if:

- Any number of players deviate arbitrarily from their program.
- The deviating players collude.

This seems like an extremely powerful theorem: practically any type of interaction among mutually suspicious parties, with any set of correctness and secrecy requirements, can be formulated in terms of a function of the local inputs of the parties, and therfore can be realized.

Remark: This is somewhat cheating, since some tasks can be modeled only as a "reactive functionality" of the inputs. These tasks, however, can be handled as well.

On further reflection, it's not really clear what the theorem is actually saying. What does it mean that the protocol is "correct" or "secure"?

Rigorously capturing the set of security requirements from such protocols has turned out to be elusive and took some time to formulate. Historically, we first had protocols that intuitively seemed to do the job, but with no meaningful definition or proof (YAO86, GMW87, BGW88, CCD88). Adequate definitions started to trickle in only later (91, 93, 95...) and it took time until proofs could actually be constructed.

In class we will follow the following procedure:

1. First sketch a definition,
2. Then sketch one construction (GMW87).

Also, for the most part we'll concentrate on the simpler case of two-party computations. As we'll see later, this simple case contains within it much of the complexity and flavor of the multi-party case.

2 Defining Two-Party Secure Computation

We saw two approaches for defining what it means for a protocol to realize some task:

- A "game based" approach that puts restrictions on any adversary in order to capture "secrecy", and "correctness".
- An "ideal model" based approach that requires constructing a "simulator" for each adversary. In this approach the "correctness" and "secrecy" requirements are blended into a single "simulation" requirement.

In the case of encryption, the two approaches gave equivalent definitions (in all the cases– symmetric, asymmetric, CPA, CCA encryptions).

In the case of commitment, the two definitions made sense but were not equivalent.

We'll see that in the case of general SFE the situation gets worse: it's not even clear how to formulate a meaningful game-based definition. The main issue is that the "secrecy" and "correctness" requirements are intertwined.

To see this, first note that we cannot require corrupted parties to use the input values they received from the outside in the protocol. In other words, we cannot guarantee that the parties which follow the protocol get $f(x_1, ...x_m)$ where $x_1...x_m$ are the external inputs. Instead, we need to allow the deviating parties to choose new values to be entered into the computation. But what should this process of choosing new values look like?

For instance, consider the function $f(x_1, x_2) = x_1 \oplus x_2$. That is, each party contributes an (a priori secret) input value and obtains the exclusive or (XOR) of the two inputes. The protocol instructs P_1 to send its input to P_2; then P_2 announces the result. Intuitively, this protocol is insecure since P_2 can unilaterally determine the output, after learningP_1's input – rather than having to decide whether to learn P_2's input OR to influence its output. Yet, the protocol maintains secrecy (which holds vacuously for this problem since each party can infer the input of the other party from its own input and the function value), and is certainly "correct" in the sense that the output fits the input that P_1 "contributes" to the computation. To rule out such protocols, we require that the bad parties choose their inputs without any knowledge of the inputs of the good parties. This means that the approach depends on secrecy, and secrecy depends on correctness.

There are more examples that bring up different issues. But this issue alone already implies that we cannot separately require correctness and privacy. So, instead we'll go with the alternative approach, namely by simulation of an ideal system that has a trusted party. Here we'll proceed as follows:

A two-party function f is a randomized function $f : N \times D^2 \times R \to D^2$ for some domain D. (Here R is taken to be the random input of the function.)

We define a protocol that securely realizes a function f in three steps:

- Executing the protocol: E (environment),A (adversary),π (protocol)

  - Participants: $E, A, \pi = (\pi_1, \pi_2)$
  - E gives inputs
  - A decides who to corrupt, tells E
  - π and A interact
  - π and A give outputs to E
  - E outputs a bit

- Ideal process: E, S, TP_f
  - Participants: E, S, dummy parties D_1, D_2 and TP_f
  - E gives inputs
  - S decides who to corrupt, tells E
  - D_1, D_2 give inputs to TP_f (corrupted party gives input only if S says to do so)
  - TP_f chooses a random $r \leftarrow R$, computes $f(n, x_1, x_2, r)$ gives outputs to D_1, D_2, S
  - D_1, D_2, S give outputs to E (Only if S says to do so)
  - E outputs a bit

- Realize:

  A protocol π securely realizes a two-party function f if for any polynomial time adversary A there exists a polynomial time adversary S such that for any polynomial time environment we have:

$$REAL_{\pi,A,E} \sim IDEAL_{TP_f,S,E}$$

Guarantees:

- Secrecy
- Correctness
- Input Independence Taken together, these items constitute the achievement of secure computation.

Note: The ideal model allows the adversary to not give output, and even to prevent the good party from getting output after the deviating party learned the output. There is indeed no way to force a deviating party to participate. The issue of learning the output without letting the other party learn his is more subtle. This issue is often called "fairness" and is out of scope for this course.

3 Construction for Semi-Honest Parties

We'll start with a very simple case, where both parties follow the protocol (including making all the right random choices). Here we only want to protect against adversaries that want to learn information by looking at the communication. We'll call such parties "Semi-Honest".

3.1 Oblivious Transfer

A main ingredient in this construction is a task called Oblivious Transfer. Here there is a sender T that has l inputs $t_1, ..., t_l$ and a receiver R with input $r \in l$. The specification is that R should learn t_r and nothing else, whereas S should learn nothing (in particular nothing about r).

Using our formalism, an $OT(l, m)$ protocol is a protocol that securely realizes the function $OT((t_1, ..., t_l), r) = (-, t_r)$, where each $t_i \in M$. This primitive was first suggested by Rabin in '81 as a different variant and later in EGL82.

How is this realized? Recall, that we are only interested in semi-honest parties. For simplicity, let's describe the solution for a 1-out-of-2 binary OT (i.e. $l = 2$, $M = \{0, 1\}$):

The EGL Protocol:

Let $F = (G, S, F, F^{-1})$ be a TDP, and let B be a Hamiltonian Circuit Problem for F.

1. T (on input s_0, s_1) runs $(i, t) \leftarrow G(1^n)$, sends i to R.
2. R (on input r) chooses $x_0, x_1 \leftarrow S(i)$, computes $y_r = F(i, x_r), y_{1-r} = x_{1-r}$, sends y_0, y_1 to S.
3. T computes $b_0 = t_0 + B(F^{-1}(t, y_0)$, sends b_0, b_1 to R.
4. R computes $s_r = b_r + B(x_r)$.

Theorem: The EGL protocol securely realizes OT.

Proof:

- Correctness is straightforward.
- R's input r is perfectly hidden, since S's view is independent of r.
- t_{1-r} is hidden due to the fact that B is a HCP of F.

More formally, we construct a simulator S. S will be placed in between the adversary A. It will run an instance of A and pass to it the input received from E.

Now we have two possible cases:

- CASE #1: A corrupts T:
  - S corrupts T and hears t_0, t_1.

- S generates three messages in place of A and sends them to E:
  - A pair $i, t \leftarrow G(1^n)$
  - A message i that will be sent as the first message.
  - A message $y_0, y_1 \leftarrow G(i)$ that will be sent as the second message.
  - A message computed by the protocol: $b_j = t_0 + B(F^{-1}(t, y_j))$ that will be sent as the third message.

  The claim is that the simulated view is distributed identically to the real one.

- CASE #2: A corrupts R:
  - S corrupts R and hears r.
  - S first generates for A two messages and sends them to E:
    - A message of an index $i \leftarrow G(1^n)$ that will be sent as the first message (notice that here t is NOT given to E).
    - Values $x_0, x_1 \leftarrow S(i)$.
    - A message of y_0, y_1 where $y_r = F(i, x_r),\ y_{1-r} = x_{1-r}$ that will be sent as the second message.
  - After R receives from TP_{OT} the output bit a, it generates an additional message of b_0, b_1, where $b_r = a + B(x_r)$, and b_{1-r} is random. This message will be sent as the third message.

The claim is that the simulated view is identical to the real one, except that the bit b_{1-r} is random rather than $t_{1-r} + B(F^{-1}(t, x_{1-r}))$. Therefore a distinguishing environment can be turned into a predictor that contradicts the security of the HCP B.)

3.2 General two party SFE given OT

Let f be a two party function. Recall that f has three inputs - the first input is the security parameter and is known to both parties, the second and third are known to the first and second party, correspondingly.

We'll represent f as an arithmetic circuit over GF_2 with addition and multiplication gates, with $3 * l$ input wires, such that P_0 has the first l input wires, P_1 has the next l, and the last l are the random input. Similarly, there are $3 * l$ output wires where the first half describes the output of P_0 and the second half the output of P_1. Potentially, there is a different circuit for each value of the security parameter.

Assume each P_i has input $x_1^i ... x_l^i$. The overall plan:

1. P_0 shares its input with P_1: for each j, choose some random $s_(0, j)$, $s_(1, j)$ such that $s_(0, j) + s_(1, j) = x_(0, j)$, sends $s_(1, j)$ to P_1 and keeps $s_(0, j)$. For each random input wire, P_0 chooses a random share $s_(j, 0)$.

   P_1 does the same.

2. The parties evaluate the circuit, gate by gate, keeping the following invariant:

   For each wire w in the circuit, the parties keep random shares $s_(0, w)$, $s_(1, w)$ such that $s_(0, w) + s_(1, w) = v_w$, where v_w is the value of the wire w in the evaluation of the circuit on inputs $\vec{x}$.

3. In the end, each party reveals its share of the output wires that correspond to the output of the other party. It then computes its own output and outputs it.

   It remains to describe how to implement step (2), namely the evaluation of the circuit. We have two cases:

   The gate is an addition gate. That is, we have two input wires, and each party has a share of each wire.

   P_0 has s_{u_0}, s_{v_0}, needs to compute s_{w_0}

   P_1 has s_{u_1}, s_{v_1}, needs to compute s_{w_1}

   s.t. $s_{w_0} + s_{w_1} = (s_{u_0} + s_{u_1}) + (s_{v_0} + s_{v_1})$

So, it's enough that each P_i will locally compute $s_{w_i} = s_{u_i} + s_{v_i}$. (Indeed, the invariant will be preserved.)

The gate is a multiplication gate. That is:
P_0 has s_{u_0}, s_{v_0}, needs to compute s_{w_0}
P_1 has s_{u_1}, s_{v_1}, needs to compute s_{w_1}
s.t. $s_{w_0} * s_{w_1} = (s_{u_0} + s_{u_1}) * (s_{v_0} + s_{v_1})$

Here we will use OT. That is, one of the parties (say, P_0) will choose a random value to bit its share s_{w_0}, and will allow P_1 to learn the other share via OT. More precisely:

1. P_0 chooses a random s_{w_0}.

2. P_0, P_1 engage in an 1-out-of-4 OT on {0,1} with the following inputs:

   Inputs for the sender (P_0):

   $O_{(0,0)} = s_{w_0} + (s_{u_0} + 0) * (s_{v_0} + 0)$

   $O_{(0,1)} = s_{w_0} + (s_{u_0} + 0) * (s_{v_0} + 1)$

   $O_{(1,0)} = s_{w_0} + (s_{u_0} + 1) * (s_{v_0} + 0)$

   $O_{(1,1)} = s_{w_0} + (s_{u_0} + 1) * (s_{v_0} + 1)$

   Input for the receiver (P_1):

   (s_{u_1}, s_{v_1})

   That is, the output of P_1 will be

   $O_{(s_{u_1}, s_{v_1})} = s_{w_0} + (s_{u_0} + s_{u_1}) * (s_{v_0} + s_{v_1})$

   Which is correct. Furthermore, each share is random, and in addition no party has learned anything in the process other than its share.

Theorem: Assuming that the OT protocol in use securely realizes OT, the above protocol securely realizes f.

On the proof: While the intuition for the security of the protocol is clear, Actually proving is is not trivial. The main difficulty here is the use of the OT protocol: How do we "reduce" the security of the general function evaluation protocol to the security of the OT?

A natural way to do this would be in two steps:

1. Formulate and analyze the function evaluation protocol as a protocol that uses an "ideal OT".
2. Show that "composing" such a protocol with a protocol $\pi_O T$ that securely realizes OT results in a secure function evaluation protocol in the standard setting.

This is indeed the way we do it, but in order to do so one has to build some machinery:

- What does it mean for a protocol to use an "ideal OT" (or, in general an "ideal subroutine")
- A general "composition theorem" that makes good of the above plan.

This is outside the scope of this class and will be discussed next semester.

3.3 Dealing with malicious faults

In the presence of general faults (also called malicious faults, or Byzantine faults), the protocol for semi-honest parties looks "ridiculously naive", in that it trivially breaks down if the parties deviate even slightly from the protocol. Still, we'll show that it can be transformed or "hardened" into a protocol that withstands malicious faults. In fact, we'll see a very general result:

Theorem: Assume one way functions exist. Then, there exists a general transformation T such that, for any two party function f and for any polytime protocol π that securely realizes f for semi-honest parties, the protocol $T(\pi)$ securely realizes f for general (Byzantine) faults.

The transformation: The idea is to make use of the power of Zero Knowledge proofs. That is, we'll have each party prove, in each step, that the message it just sent is in accordance with the protocol, its secret input, and the messages that it receives so far. Since the computation required to generate each message is polytime, the said statement is an NP statement.

There is a snag here, however: The internal computations of parties are not just a function of the secret input and the incoming messages. There is also the secret, local random choices of each party...

To deal with that, we'll have each party "commit to its random tape" at the beginning of the computation. Later, at each step, the party will prove that is actually used the random choices from the committed random tape. This will make sure that:

- the actions of each party are uniquely determined given its secret input and the transcript of communication so far
- the random choices of each party remain secret, essentially as in the semi-honest protocol.

More precisely, the transformed protocol $T(\pi)$ proceeds as follows. Let (COM,VER) be a commitment scheme, and let ZK be a ZK proof system for language in NP. (Both can be constructed based on OWFs.)

1. Commitment to random tape of P_0:
   - P_0 chooses r_1^0,....r_k^0, $\rho_1 ... \rho_k$, and sends c_1^0-COM(r_1, ρ_1),....,COM(r_k,ρ_k) to P_1.
     (here k=poly(n,l), and ρ_i is the randomness used for the ith commitment. For simplicity we assume that COM is non-interactive ,i.e. one message. But the same works even for interactive commitment.)
   - P_1 chooses $r_1^1 ... r_k^1$ and sends to P_0

   [The "committed random tape" of P_0 is defined to be $r_1 ... r_k$ where $r_i = r_i^0 + r_i^1$]

   Commitment to random tape of P_1: Done similarly
2. The parties run protocol π, with the following differences:
   - The parties use the committed random tapes for their random choices
   - After sending each message m, the sending party proves in ZK that the message m is the result of running the prescribed protocol on some input x and some random input r, and given the communication so far.

We assume that the sending party is P_0. Then given protocol π and transcript τ, let L be the following language:

Let string $\vec{m}_j$ be made up of the first j messages sent by the original protocol π, and let string $\vec{c}$ be the commitments sent by P_0 in the commitment state. Then let $L_{\vec{m}_j,\vec{c}} = \{m | \exists x, r_1^0 ... r_k^0, \rho_1^0 ... \rho_k^0$ such that

1. c_i=$c_i = COM(r_i^0, \rho_i^0)$ $\forall i = 1...k$
2. m is the message sent by pi given input x, random input $\vec{r}$, and incoming messages $\vec{m}_j\}$.

   Then, P_0,P_1 engage in a ZK protocol for whether $m \in L_{\vec{m}_j,\vec{c}}$, where τ is the transcript (of π) so far and $c_1 ... c_k$ are the commitments from step 1. If the verification fails, the protocol is aborted.

Analysis:

We will not do it here. First analyzes were quite hairy. But, if one uses composability, then the analysis becomes much more reasonable.

3.4 The multi-party case

The general principles are the same. However, things become (even) more complicated, both on a definitional level and on the protocol level. Some highlights:

- The specific properties of the communication channels become crucial:
  - synchrony
  - authenticity
  - atomic broadcast
- Have to deal with coalitions of deviating parties, both from a definitional aspect and from a protocol aspect.
- Composability becomes harder, since different sets of parties may run subprotocols without knowing of each other.

Nevertheless, it can be done.