COM S 6830 - Cryptography November 19, 2009

Lecture 22:Multiparty secure computation and oblivious transfers

Instructor: Rafael Pass *Scribe: Srivatsan Ravi*

1 Secure multiparty computation-Review

Consider a set of parties $P_1.....P_n$ with private inputs $x_1.....x_n$ that want to compute a function $f(x_1, ...x_n)$ which outputs a single public value. If a trusted external party existed, all parties would give their output to this party which would compute the function f and publish this output. This is the IDEAL model where the output is consistent with the inputs x_i and function f and nothing about the private inputs is revealed beyond whatever information is contained in the public output(privacy). In the absence of a trusted external party, the parties engage in a protocol among themselves called the REAL model. In this case, a malicious party may follow an arbitarily feasible strategy implementable by non-uniform expected poly-time machines. Secure computation deals with the problem of emulating the IDEAL model in the REAL model.

Definition 1. *Protocol* Π *is said to securely compute f if for all non-uniform PPT machines A in the REAL model and non-uniform PPT machines A' in IDEAL model such that*
$\forall z, \forall n \epsilon N, \forall \vec{x} \epsilon (\{0,1\}^n)^{m(n)}$

$$\text{REAL}(A(z), \vec{x}) \approx \text{IDEAL}(A'(z), \vec{x})$$

i.e for every PPT adversary controlling a subset of parties in the REAL model, there exists a PPT simulator controlling the same subset of parties in the IDEAL model such that the output of all parties in the REAL model is computationally indistinguishable from their outputs in the IDEAL model

Definition 2. *Let* f: $(\{0,1\}^m)^n \rightarrow (\{0,1\}^m)^n$ *be a poly-time computable function, and let t be less than n/2. Assuming the existence of trapdoor permutations,there exists an efficient n-party protocol that securely computes f in the presence of up to t corrupted parties.*

Definition 3. *Honest but curious(HBC) adversary as one that follows the protocol exactly as it is defined, yet attempts later to find additional information.*

This type of honesty helps us construct secure multiparty protocols easier. This maybe enforced through the use of coin-tossing protocols and ZK proofs of knowledge which allow parties to prove that they are following the protocol honestly. This adversary model will be used to establish the notion of Oblivious transfer(OT)

2 Oblivious transfer(OT)

Oblivious transfer protocol(OT) is a protocol by which a sender sends some information to the receiver, but remains oblivious to what is received. More specifically a $1/k$ − oblivious transfer protocol is a secure computation for party A to learn one of the k secret bits held by party B without B learning which secret A obtained. In this lecture, we concentrate on a $1/4$ oblivious transfer.

Definition 4. *1/4 Oblivious transfer: A has 4 bits* $\{a_1, a_2, a_3, a_4\}$ *and B has an integer* $b \epsilon \{1, 4\}$. *Compute the function*
$\mathrm{OT}_{1/4}(a_1, a_2, a_3, a_4, b) = (\perp, a_b)$

Theorem 5. *Assume the existence of trapdoor permutations* $\{f_i\}_{i \epsilon I}$ *over* $\{0, 1\}^n$. *Then there exists a HBC secure computation for* F_{OT}. *Let* h *be a hardcore predicate for any function from the family*

A:	$i, t \leftarrow \mathrm{Gen}(1^n)$
$A \rightarrow B$*:*	i
B:	*if* $j \neq b$, *then* $y_j \leftarrow \{0,1\}^n$ else $x \leftarrow \{0,1\}^n$; $y_j \leftarrow f_i(x)$
$B \rightarrow A$*:*	y_1, y_2, y_3, y_4
A:	$z_j = h(f^{-1}(y_j)) \oplus a_j$
$A \rightarrow B$*:*	z_1, z_2, z_3, z_4
B:	*Output* $h(x) \oplus z_b$

PROOF Outline:

- Party A learns nothing new in this protocol because each y_j it receives are independent of the integer b and are uniformly distributed.
- Party B learns nothing more than a_b since being a honest but curious adversary, he knows only one of the values of $f^{-1}(y_j)$. Consider a simulator S which receives some q_b as output, it follows the protocol and chooses i, t and invokes B and hands over i as thought it was from A. After receiving y_1, y_2, y_3, y_4 from B,S computes $z_i = h(x) \oplus q_b$ and hands over each z_i to B. S outputs whatever B outputs. Now clearly whether S handed B the correct z_i cannot be detected in poly-time otherwise h would not be hard-core

3 Security in HBC adversary model

In this section, we describe the construction of secure two-party computation in the HBC adversarial model.

- Let f be the two-party functionality that is to be securely computed. Then, the parties are given an arithmetic circuit over the finite field GF(2) that computes the function f .
- Such an arithmetic circuit over a field F represents a directed acyclic graph where every node with in-degree zero represents a variable or finite element in F. The other nodes represent the SUM and PRODUCT gates

3.1 High level outline

- The protocol starts with the parties sharing their inputs with each other using a XOR scheme
- Following this, they both hold shares of the input lines of the circuit i.e, for each input i, party A holds a value a_i and party B holds a value b_i , such that both a_i and b_i are random under the constraint that $(a_i + b_i)$ equals the value of the input into this circuit line.
- Evaluate the circuit gate-by-gate
- Reveal all shares

3.2 Detailed description

There are three gates that we consider for the circuit- the NOT, XOR and AND gate. Clearly NOT gate represents g(x)=x+1 in GF(2) while the AND gate represents multiplication. Additionally, the XOR gate stands for g(x,y)=x+y

SUM gate:

Assume some party P_1 holds values a_i and a_j and P_2 holds the values b_i and b_j. Each party can just locally add its values. $(a_i + a_j) + (b_i + b_j) = (a_i + b_i) + (a_j + b_j)$. Since no information is transferred in evaluating the gate, neither party learns anything more than it already knew

AND (PRODUCT)gate:

We need to compute random shares c_i and c_j such that
$c_i + c_j = a.b = (a_i + a_j)(b_i + b_j) = a_i b_i + a_i b_j + a_j b_i + a_j b_j$

We consider the case when the split is done for only two shares to make the analysis simpler. The idea is that input wires to gate have values a and b. Party P_i has the shares a_i and b_i. Party P_j has the shares a_j and b_j . Output of P_i is c_i and P_j's is c_j.
Goal is that P_i computes c_i knowing its share values and without knowing the shares of P_j

a_j	b_j	Receiver output
0	0	0
0	1	c_i
1	0	c_j
1	1	$(c_i + c_j)$

P_1 chooses a random bit $r \epsilon_R \{0, 1\}$ and sets this value to its share of output line and defines the following table for computing the receiver output

a_j	b_j	Receiver output
0	0	r
0	1	$c_i \oplus r$
1	0	$c_j \oplus r$
1	1	$(c_i + c_j) \oplus r$

Now the parties use $\mathrm{OT}_{1/4}$. P_1 is the sender and inputs the message i in row i of the above table. P_2 which is the receiver now sets its input i as defined in the table. On receiving the output from $\mathrm{OT}_{1/4}$, P_2 sets c_j to this output and this becomes its share of the output line of the gate. P_1 has already set its output to the value of the random bit it chose.

If a certain output gate provides a bit of P_i's input, then P_2 will reveal its share of this output line to P_1. In this manner, each party reveals its share of the output line to the other party

4 Security against malicious adversaries

In this section,we focus on how to get malicious security. One idea could be to use ZK to get each party to prove it is doing what it is supposed to do. We need for force malicious adversaries to behave like honest-but-curious adversaries. This is done relative to the given input and a uniformly chosen random tape.

- Such a COMPILER begins by having each party commit to its input. In order to ensure the committing party knows the value being commited to, usually this is also followed by a ZK-POK(Zero knowledge prootf of knowledge)

- Generating randomness:Parties run a coin-tossing protocol for their random tape
- Regular coin-tossing protocols will not suffice it is important that the parties random tapes remain secret. Hence, if the parties receive the same uniform distribution, it is not sufficient
- The coin-tossing protocol will be implemented with commitments. Hence one party will receive a commitment to a uniformly distributed bit-string and the other receives the string and the de-commitment for later verification

The next lecture will cover the above topic in formal detail