6.876/18.426: Advanced Cryptography 5/5/03

Lecture 20: Secure Multi-Party Computation in the HBC Model

Scribed by: Yael Tauman

1 Overview

In this lecture, we discuss the problem of constructing a secure multi-party protocol for computing a function whose input is divided among a number of parties. We want such a protocol to compute the function "correctly" and "privately", even when some of the parties are malicious and deviate arbitrarily from the protocol (in a probabilistic polynomial-time manner).

A first question one may ask is: Do such protocols exist? It turns out that if trapdoor permutations exist and if less than half of the parties are malicious, then every function has a *correct* fault-tolerant protocol that offers the maximum degree of *privacy*. The complexity of the protocol is polynomial in the time-complexity of the function. Furthermore, there exists a polynomial-time algorithm that on input a Turing-Machine specification of the function outputs such a protocol.

In this lecture, rather than diving into the formal definitions of the above requirements (*correctness* and *privacy*), we will focus on constructions. However, before doing so, let us illustrate the intuitive meaning of these requirements.

Consider an environment in which in addition to the participating parties there is a *trusted party*. In the "trusted party environment", each party sends its input to the trusted party using a *secure* channel; then the trusted party computes the output and sends it to all the parties. Since the trusted party is non-faulty, the computed value of the function is the correct one (with respect to the inputs given to the trusted party). It is also clear that no collaboration of faulty parties can learn more than what can be computed from the final output and their local inputs.

We would like a multi-party protocol that would have the same *correctness* and *privacy* properties as a computation in the "trusted party environment".

2 The Main Result

We assume, without loss of generality, that all parties compute the same output function f. We can create such a function from the local output functions $f_1, \ldots, f_n$, by letting $f(x_1, \ldots, x_n) \triangleq E_1(f_1(x_1, \ldots, x_n)) \circ \ldots, \circ E_n(f_n(x_1, \ldots, x_n))$, where E_i is a secure public-key encryption for which only P_i knows the decryption key. We also assume, without loss of generality, that the function f is binary (into $\{0,1\}$). Our general goal is to prove the following theorem.

Theorem 1. *Let $f : (\{0,1\}^m)^n \to \{0,1\}$ be a polynomial-time computable function, and let $t < \frac{n}{2}$. If there exists a trapdoor permutation then there exists an efficient protocol for n parties $P_1, \ldots, P_n$, that privately computes f in the presence of t faulty parties.*

Generally speaking, we will prove this theorem in two parts. First, we will present a protocol π that privately computes f if all parties are "honest but curious". Then, we will present a method of "compiling" π into another protocol, $C(\pi)$, that privately computes f in the presence of $t < \frac{n}{2}$ faulty parties.

In this lecture we will concentrate on the first part. Namely, we will assume that all parties are honest but curious (HBC).

2.1 The HBC Model

In the HBC model, we assume that all parties are honest but curious. That is, we assume that each party executes the original protocol but tries to compute as much additional information as possible. More specifically, we assume that each party uses for its random tape the output of random coin tosses, and that in each communication round it sends exactly the message as instructed in the protocol (the protocol defines a unique message to be sent, as a function of the initial input, the random tape, and the communication received so far). Moreover, we assume that the parties don't listen to any communication other than those sent to them. So, the parties may deviate from the protocol only in their internal computation, but the messages they send are in accordance with the protocol.

In the remainder of this lecture we will prove the following theorem.

Theorem 2. *For any polynomial-time computable function $f : (\{0,1\}^m)^n \to \{0,1\}$ there exists an efficient n-party protocol that privately computes f in the HBC model.*

Actually, we will show that there exists a compiler $\mathcal{P}$ such that for any function f, $\mathcal{P}(f)$ outputs n algorithms $(H_1, \ldots, H_n)$, which form a protocol that privately compute f.

Proof. $\mathcal{P}$, on input a Turing-Machine M_f, which computes some function $f : (\{0,1\}^m)^n \to \{0,1\}$, transforms M_f into a circuit C_f that computes f. Note that since f is polynomial-time computable there exists a Boolean circuit of polynomial-size that computes f. We will assume, without loss of generality, that C_f consists only of $\neg$ and $\wedge$ gates ($a \vee b = \neg a \wedge \neg b$), and that the maximum fan-in of a gate is two. We consider these operations as operations over the field $\mathbb{F}_2$. The $\neg$ operator is replaced with the function $g(x) = x + 1$ and the $\wedge$ operator is replaced with the multiplication function $g(x, y) = x \cdot y$.

$\mathcal{P}$, given C_f, will output $(H_1, \ldots, H_n)$, which are defined as follows.

1. Each H_i shares each of his input bits with all other parties, in a way that exactly n parties are needed in order to gain any information about the input bit. More specifically, H_i will share a bit b by choosing at random n bits $b_1, \ldots, b_n$ satisfying $b = \sum_{i=1}^{n} b_i$, and sending party j the bit b_j.

   At this point, the parties hold *random* shares allowing them to obtain the values of each input wire of the Boolean circuit. Our purpose is that the parties will hold *random* shares allowing them to obtain the value of the output wire of the circuit. To this end, the parties will scan the circuit from the input gates to the output gate. For each gate, they will generate random shares of the value of its output wire from the random shares of the values of its input wires. This is done as follows.

2. For each $\neg$ gate, the value of its output wire is obtained by adding the constant 1 to the value of its input wire. We assume (by induction) that the value, b, of its input wire is randomly distributed among all parties, as follows: $b = \sum_{i=1}^{n} b_i$, where party i holds the share b_i.

   Thus, we will distribute the value of the output wire of $\neg$ as follows: one of the parties, say the first party, will calculate his share by adding the constant 1 to his share of the input wire: $b_1 + 1$. All other parties let their share of the output wire be equal to their share of the input wire. Thus, we obtain the output value $(b_1 + 1) + b_2 + \ldots + b_n = \sum_{i=1}^{n} b_i + 1 = b + 1$, as desired. Also note that these shares remain random.

3. For each $\wedge$ gate, the value of its output wire is obtained by multiplying the values of its two input wires. Denote these values by c and d. Assume (by induction) that these two values are randomly shared among all parties as follows: $c = \sum_{i=1}^{n} c_i$ and $d = \sum_{i=1}^{n} d_i$, where party i holds the shares c_i and d_i. The parties need to compute n random shares $\{b_i\}_{i=1}^{n}$, such that

$$b \triangleq \sum_{i=1}^{n} b_i = (\sum_{i=1}^{n} c_i)(\sum_{i=1}^{n} d_i) = (\sum_{i=1}^{n} c_i d_i) + \sum_{1 \leq i < j \leq n} (c_i d_j + c_j d_i),$$

   and such that each party P_i knows only b_i. Define $b_{i,j}$ so that for every i, $b_{i,i} = c_i d_i$ and for every $i \neq j$, $b_{i,j} + b_{j,i} = c_i d_j + c_j d_i$, and let $b_i = \sum_{j=1}^{n} b_{i,j}$. Note that

$$\begin{aligned} c \cdot d = \quad & (\sum_{i=1}^{n} c_i)(\sum_{i=1}^{n} d_i) = \\ & (\sum_{i=1}^{n} c_i d_i) + \sum_{1 \leq i < j \leq n} (c_i d_j + c_j d_i) = \\ & \sum_{i=1}^{n} b_{i,i} + \sum_{1 \leq i < j \leq n} (b_{i,j} + b_{j,i}) = \\ & \sum_{i=1}^{n} \sum_{j=1}^{n} b_{i,j} = \sum_{i=1}^{n} b_i = b. \end{aligned}$$

   Also note that for every i, $b_{i,i} = c_i \cdot d_i$ is uniformly distributed (as c_i and d_i are uniformly distributed) and can be compute by party P_i. The question is how can two parties, P_i and P_j, privately compute $b_{i,j}$ and $b_{j,i}$, so that the resulting shares will be uniformly distributed among all possible shares? This will be done using a $\binom{4}{1}$ Oblivious Transfer[1], as follows.

   Note that $b_{i,j} + b_{j,i} = c_i d_j + c_j d_i$ implies that $b_{i,j} = c_i d_j + c_j d_i + b_{j,i}$ (since the computation is in $\mathbb{F}_2$). Thus, P_j will pick $b_{j,i}$ at random and will transfer to P_i, using a $\binom{4}{1}$ Oblivious Transfer, one of the following values: $b_{j,i}$, $b_{j,i} + d_j$, $b_{j,i} + c_j$ or $b_{j,i} + c_j + d_j$. P_i will choose to receive the first value $(b_{j,i})$ if $c_i = d_i = 0$, to receive the second value $(b_{j,i} + d_j)$ if $c_i = 1$ and $d_i = 0$, to receive the third value $(b_{j,i} + c_j)$ if $c_i = 0$ and $d_i = 1$, and to receive the forth value $(b_{j,i} + c_j + d_j)$ if $c_i = d_i = 1$. P_i will set $b_{i,j}$ to be equal to the value received. Note that this sub-protocol satisfies that $b_{i,j} + b_{j,i} = c_i d_j + c_j d_i$, and that $b_{i,j}$ and $b_{j,i}$ are both uniformly distributed. Thus, the resulting shares $\{b_i\}_{i=1}^{n}$ are also uniformly distributed among all shares, as desired.

□

Notice that the above construction relies heavily on the fact that every function consists of a sequence of local computations. Also note that the above protocol is information

[1] Recall that a construction of a $\binom{4}{1}$ Oblivious Transfer was given in the previous lecture.

theoretically private, except for the $\binom{4}{1}$ Oblivious Transfer protocol which is only computationally private. An interesting question one may ask is whether there exists a $\binom{4}{1}$ Oblivious Transfer protocol which is information theoretically private? Or can we achieve an information theoretically private multi-party function evaluation in any other way (in the HBC model)?