Cryptographic Protocols

Notes for Lecture 8

Scribe: Sandro Coretti

About the notes: These notes serve as written reference for the topics not covered by the papers that are handed out during the lecture. The material contained therein is thus a strict subset of what is relevant for the final exam.

This week, the notes treat sharing schemes, an MPC protocol that is information-theoretically secure against a passive adversary corrupting up to $t < n/2$ of the players, and a proof that there exist no information-theoretically secure protocols for $t \geq n/2$.

8.1 Sharing Schemes

A $t$-out-of-$n$ secret-sharing scheme allows an honest dealer $D$ to distribute a secret $s$ among $n$ players, such that any subset of $t$ players has no information about $s$, but every set of $t + 1$ players can collaboratively reconstruct the secret. The most famous secret-sharing scheme is Shamir’s Sharing Scheme [Sha79] (cf. Section 8.1.2). It uses polynomials to obtain the desired properties. Before presenting the scheme, we first take a look at Lagrange interpolation.

8.1.1 Lagrange Interpolation

Consider a field $\mathbb{F}$ and $n$ points $(\alpha_1, s_1), \ldots, (\alpha_n, s_n)$ (for distinct $\alpha_i$). Lagrange interpolation can be used to find the unique polynomial over $\mathbb{F}$ of degree at most $n - 1$ that contains these points: For $i = 1, \ldots, n$, consider the polynomial

$$\ell_i(x) = \prod_{j=1}^{n} \frac{x - \alpha_j}{\alpha_i - \alpha_j}.$$ 

Clearly, $\ell_i(\alpha_i) = 1$ and $\ell_i(\alpha_j) = 0$ for $j \neq i$. Therefore, a polynomial that goes through the points $(\alpha_i, s_i)$ for $i = 1, \ldots, n$ is given by

$$g(x) = \sum_{i=1}^{n} \ell_i(x) \cdot s_i.$$ 

The polynomial $g$ can be shown to be unique (cf. Exercise 8.1.a)). Moreover, note that the value of $g$ at some fixed position is a linear function of the $s_i$. 
8.1.2 Shamir’s Sharing Scheme

Consider a finite field $\mathbb{F}$ of size greater than $n$. Assume there is a unique, fixed, and known value $\alpha_i \in \mathbb{F} \setminus \{0\}$ associated with every player $P_i$. The dealer $D$ chooses a random polynomial $p(\cdot)$ over $\mathbb{F}$ and of degree at most $t$ such that $p(0) = s$. This can be done by choosing $r_1, \ldots, r_t$ at random from $\mathbb{F}$ and defining

$$p(x) := s + r_1 x + r_2 x^2 + \ldots + r_t x^t.$$ 

Then, for $i = 1, \ldots, n$, he sends the value $s_i := p(\alpha_i)$, called the $i$th share, to player $P_i$. All shares together are called a sharing of $s$. Sometimes, we denote a sharing of $s$ by $[s]$.

Given any $t+1$ shares, one can compute the secret $s$ using Lagrange interpolation. Furthermore, one can show that up to $t$ shares give no information about $s$ (cf. Exercise 8.1.b)).

Shamir sharings are linear. That is, the sharing $s = (s_1, \ldots, s_n)^T$ can be obtained by multiplying the vector $r = (s, r_1, \ldots, r_t)^T$ consisting of the secret and the random coefficients by a matrix $M$ (which the reader can find as an exercise). That is, $s = Mr$. From two sharings $s = Mr$ of $s$ and $s' = Mr'$ of $s'$, a sharing of the sum $s + s'$ can be computed by having each player adding his shares. This is due to the fact that $s + s' = Mr + Mr' = M(r + r')$ and thus the secret in $s + s'$ is $s + s'$ (since $M$ is injective).

8.2 Passively Secure Multi-Party Computation

In this section we present a simplified version of the MPC protocol by [BGW88], which provides security against up to $t < n/2$ passively corrupted players. This is optimal in the sense that there cannot exist a protocol with these properties that allows to evaluate an arbitrary function and tolerates more passively corrupted players (cf. Section 8.3).

The function that the players want to evaluate must be given as an arithmetic circuit over a finite field $\mathbb{F}$ of size $q > n$, consisting of addition and multiplication gates. We will denote the player set by $P = \{P_1, \ldots, P_n\}$. Without loss of generality we assume that player $P_i$ holds input $x_i \in \mathbb{F}$.

The basic idea of the protocol is the following: At the beginning, every player shares his input using Shamir’s sharing scheme. Then, in a gate-by-gate fashion, the players will evaluate every gate. Here, evaluate means that if the inputs to some gate are shared among the players, then, after the evaluation of the gate, the output of the gate is shared among the players. Once the last gate is evaluated, the players reconstruct the output using Lagrange interpolation. The full protocol is depicted in Figure 1.

Since it is clear from the description given in Section 8.1 how the sharing and reconstruction phases work, we now focus on how to evaluate addition and multiplication gates.

Evaluating addition gates is easy as already pointed out in Section 8.1.2: Each player just adds his shares of the summands to obtain a share of the sum. This can obviously be done without communication. Moreover, it easy to see that this argument extends to any linear function, not just addition.

Evaluating multiplication gates is somewhat more challenging. Suppose that from two sharings $[a] = (a_1, \ldots, a_n)$ and $[b] = (b_1, \ldots, b_n)$ the players wish to compute a sharing $[c] = (c_1, \ldots, c_n)$ of the product $c = a \cdot b$. Following the same approach as with addition, one might be tempted to simply have each player $P_i$ compute the product $d_i = a_i \cdot b_i$ of his two shares. However, now the

\footnote{Recall that $n$ denotes the number of players.}
Passively Secure MPC

Input
To share his input $x_i$, each player $P_i$ chooses random polynomial $f(x)$ of degree at most $t$ such that $f(0) = x_i$. Then, for $j = 1, \ldots, n$, he sends $x_{ij} := f(\alpha_j)$ to $P_j$.

Addition Gates (and Linear Functions)
Each player $P_i$ starts out with the shares $a_i$ and $b_i$ of the inputs to the gate $a$ and $b$, respectively. He computes a share $c_i$ of $c = a + b$ by adding his shares, i.e., $c_i = a_i + b_i$.

Multiplication Gates
Each player $P_i$ starts out with the shares $a_i$ and $b_i$ of the inputs to the gate $a$ and $b$, respectively. He computes a share $c_i$ of $c = a \cdot b$ by as follows:
1. Compute $d_i := a_i \cdot b_i$.
2. Compute a Shamir sharing $(d_{i1}, \ldots, d_{in})$ of $d_i$.
3. For $j = 1, \ldots, n$: Send share $d_{ij}$ to player $P_j$.
   For $j = 1, \ldots, n$: Let $d_{ji}$ be share of $d_j = a_j \cdot b_j$ received from player $P_j$.
4. Compute share of $c = a \cdot b$ as
\[
c_i := \sum_{j=1}^{n} w_j d_{ji} \quad \text{where} \quad w_j = \prod_{\substack{k=1 \atop k \neq j}}^{n} \frac{\alpha_k}{\alpha_k - \alpha_j}.
\]

Output
To reconstruct a value $a$, each player $P_i$ sends his share $a_i$ to all players. Then, he uses Lagrange interpolation to compute $a$ from shares $a_j$ received by players $P_j$.

Figure 1: Passively secure MPC protocol.

values $d_i$ lie on a polynomial of degree $2t$, which implies that the players could only compute one multiplication before they would become unable to reconstruct the sharing. Moreover, the product polynomial is no longer random, which violates privacy (cf. Exercise 8.1.c)).

Therefore, we are forced to take a different approach. Let $f(x)$ and $g(x)$ denote the polynomials (of degree at most $t$) used to share $a$ and $b$, respectively. Consider the product polynomial $h(x) := f(x) \cdot g(x)$, which has degree at most $2t$. Clearly, we have $h(0) = c$ and $h(\alpha_i) = d_i$ for $i = 1, \ldots, n$. That is, since $n > 2t$, the points $(\alpha_i, d_i)$ define the polynomial $h(x)$. Hence, Lagrange interpolation (cf. Section 8.1.1) can be used to compute
\[
c = h(0) = \sum_{i=1}^{n} w_i \cdot d_i \quad \text{where} \quad w_i = \ell_i(0) = \prod_{\substack{j=1 \atop j \neq i}}^{n} \frac{\alpha_j}{\alpha_j - \alpha_i}.
\]

Thus, the product $c$ can be expressed as a linear combination $L$ of the values $d_i$. All one needs to do now is to come up with a way to evaluate $L$ on the inputs $d_i$. But we already know how to securely evaluate linear functions: Each player $P_i$ shares his input $d_i$ and applies $L$ to his shares. At this point the players have a sharing (of degree $t$) of $c$. 
Analysis

The correctness of the protocol can easily be verified since we consider passive adversaries only.

In order prove that the protocol is private, one needs to show that the adversary, until before the reconstruction phase, does not obtain any information on the inputs of uncorrupted parties or intermediate results. During reconstruction the adversary only learns information he is allowed to know according to the specification.

In the protocol there are three places in which the adversary might potentially obtain unauthorized information. It is easily seen, however, that privacy is not violated:

- When an honest party shares a value (either an input or a product share), it uses fresh randomness and thus the shares of the corrupted parties are statistically independent of the shared value.
- **Local computation** obviously does not violate privacy.
- When reconstructing, only players who are supposed to learn the output receive shares by honest players. Thus, privacy is maintained.

8.3 Impossibility for \( t \geq n/2 \)

In order to show the impossibility of information-theoretically secure MPC in the presence of an attacker corrupting more than \( n/2 \) of the parties, we first show that there exists no such protocol for the case \( t = 1 \) and \( n = 2 \); then, we reduce the general case to this base case. The proof below excludes only perfectly secure protocols, but it can be generalized to protocols with a negligible error.

Consider two parties Alice and Bob that hold values \( x_A \) and \( x_B \), respectively, and wish to compute \( f(x_A, x_B) = x_A \land x_B \). Moreover, consider an arbitrary protocol between the two. Then, the transcript of their interaction is a function \( T(x_A, r_A, x_B, r_B) \), where \( r_A \) and \( r_B \) are the random bits used by their protocols. Firstly, note that if Bob’s input is \( x_B = 0 \), then the transcript must contain no information about Alice’s input \( x_A \). This means that the random variables \( T(0, R_A, 0, r_B) \) and \( T(1, R_A, 0, r_B) \) for uniformly random \( R_A \) must be distributed identically for all \( r_B \), as otherwise Bob could infer information from \( T \). Secondly, if Bob’s input is \( x_B = 1 \), then the transcript must contain full information about Alice’s input \( x_A \). That is, the random variables \( T(0, R_A, 1, r_B) \) and \( T(1, R_A, 1, r_B) \) must have disjoint supports as otherwise Alice may not be able to infer the correct output.

Consider now a particular execution resulting in transcript \( T = T(x_A, r_A, x_B, r_B) \). To determine \( x_B \), for both \( b \in \{0, 1\} \), Alice now checks (by exhaustive search) for how many \( r_A’ \) she would have obtained \( T \) by running the protocol with \( x_A = b \), randomness \( r_A’ \), and the answers from Bob as in \( T \).² If the number is the same for both \( b \in \{0, 1\} \), then the transcript contains no information about her input, and, therefore, Bob’s input must be \( x_B = 0 \). If the number is different, then the transcript does contain information about her input, and, therefore, \( x_B = 1 \).

Consider now the general case with \( t \geq n/2 \) and assume there exists a protocol for securely evaluating any function \( f \) among the \( n \) parties. This means that there exist two sets \( M_1 \) and \( M_2 \) of tolerable passively corrupted players with \( M_1 \cup M_2 = \mathcal{P} \). W.l.o.g. assume \( P_1 \in M_1 \) and \( P_2 \in M_2 \). A protocol that allows to securely compute the function \( f(x_1, \ldots, x_n) = x_1 \land x_2 \)

---

²Alice can make that determination for each pair \((b, r_A')\) by running her protocol on input \( x_A = b \) and randomness \( r_A' \) using Bob’s answers from \( T \) until either the her protocol ends or one of her messages is different from the corresponding one in \( T \).
among the $n$ players can be used by Alice and Bob to compute the $\land$-function of their inputs: Alice simulates all players in $M_1$ and Bob those in $M_2$. This contradicts the impossibility above.

Note that the above attack is not efficient, and therefore does not rule out cryptographically secure protocols.

References
