COS 533: Advanced Cryptography — Princeton University
Lecture 7 (October 4, 2017)
Lecturer: Mark Zhandry — Scribe: Stefan Tiegel

Notes for Lecture 7

Information Theoretic Multiparty Communication

In the last lecture we saw how to construct an MPC protocol relying on the Decisional Diffie-Hellman assumption, this time we will construct an information-theoretically secure protocol. To this end, we will first introduce Shamir's Secret Sharing.

Shamir's Secret Sharing

Last lecture we saw an n-out-of-n secret sharing protocol, meaning that all n users together are able to reconstruct the message, but already $n-1$ users fail to learn anything about it. Shamir's Secret Sharing is a t-out-of-n protocol, meaning that already $t \leq n$ users together can learn the message, but any $t-1$ do not learn anything. We only give a brief description of it.

The protocol works as follows: First, fix a field $\mathbb{F}$, such that $|\mathbb{F}| > n$ and let the messages be elements of the field. For a fixed message m, the dealer now chooses a random degree $t-1$ polynomial f over $\mathbb{F}$ (by choosing its coefficients randomly, except for the constant) which satisfies $f(0) = m$. He then sets the shares to $sh_i = f(i)$. To reconstruct, t users can learn the whole function f by interpolating it and thus, also evaluate $m = f(0)$. Furthermore, it can be shown that $t-1$ users have but a negligible chance to learn m. We now use this scheme to define our MPC protocol.

The BGW Protocol (Ben-Or, Goldwasser, Wigderson)

The BGW protocol is a secure multiparty communication protocol, as long as the adversary knows the views of at most $t-1$ players, for some fixed t. As a function we would like to compute an algebraic circuit over $\mathbb{F}$ and every player has the input value of one wire. Note, that this is not a restriction from Boolean Circuits, since we can express the boolean operators as follows: $x \wedge y = x \cdot y, x \vee y = x + y - x \cdot y$, and $\neg x = 1 - x$.

At the beginning of the protocol, every player secret shares his input value to everyone else using Shamir's t-out-of-n scheme where t is the fixed value from above. Thus, after this everyone has shares for all input wires of the circuit. We now want to compute the function wire by wire. For gate g we denote its inputs by x_u and x_v and its output by x_w. The share of user i we call $sh_{u,i}$, $sh_{v,i}$, or $sh_{w,i}$ respectively. Also, let f_u, f_v, and f_w denote the functions defined by the respective shares. We now discuss how to compute the output shares of the different type of gates.

Computing + and - Gates

First, we will start with + and - gates. Note, that this also yields a way to compute multiplication by a constant. For a + or - gate we now set:

$$sh_{w,i} = sh_{u,i} \pm sh_{v,i} = f_u(i) \pm f_v(i) = (f_u \pm f_v)(i)$$

And indeed, $(f_u \pm f_v)$ has degree $t-1$ and $(f_u \pm f_v)(0) = x_u \pm x_v = x_w$ as desired. Next, we will turn our attention to $\times$ gates.

Computing $\times$ gates

As a first idea, we could try a similar approach as before: Set $sh'_{w,i} = sh_{u,i} \cdot sh_{v,i} = f_u(i) \cdot f_v(i) = (f_u \cdot f_v)(i)$. And indeed, we have $(f_u \cdot f_v)(0) = x_u \cdot x_v = x_w$, however, since both f_u and f_v are polynomials of degree $t-1$, it follows that $f_w = (f_u \cdot f_v)$ might have degree up to $2t-2$, which is too high! In this case, we would need $2t-1$ players to reconstruct and this number would grow with every multiplication gate. Thus, we would like to obtain shares of x_w using only polynomials of degree $t-1$, i.e., we would like to find a polynomial f of degree $t-1$, such that $f(0) = x_w$.

In order to do so, the players have to communicate with each other. Specifically, every user i computes his share $sh'_{w,i}$ of $(f_u \cdot f_v)(0)$ and secret shares it to everyone using Shamir's scheme with a polynomial $g_{w,i}$ of degree $t-1$ and $g_{w,i}(0) = sh'_{w,i}$. Call the share user j obtained from this $sh_{w,i,j} = g_{w_i}(j)$. We claim that this is enough to locally compute shares of x_w. To show this, we need the concept of Lagrange interpolation, a simple scheme for interpolating functions which will serve our purpose well.

Intermezzo: Lagrange Interpolation

Suppose we are given distinct elements $x_1, \ldots, x_{2t-1} \in \mathbb{F}$, then a degree $2t-2$ polynomial f over $\mathbb{F}$ can be written as follows:

$$f(x) = \sum_{k=1}^{2t-1} f(x_k) \cdot \left(\prod_{l \neq k} \frac{x - x_l}{x_k - x_l} \right)$$

One property that will be especially useful for us, is that for a given x we can write $f(x)$ as a linear combination of the $f(x_k)$.

In our setting, we can now fix $x_i = i$ and thus, have a way to represent $f_w = f_u \cdot f_v$ in the above form. Remember, that we do not necessarily care about the f_w but rather about the fact that every user can compute a share of $x_w = f_w(0)$. Lagrange interpolation allows us to write this as:

$$f_w(0) = \sum_{k=1}^{2t-1} f_w(k) \cdot \left(\prod_{l \neq k} \frac{x_l}{x_k - x_l} \right)$$

Using the fact, that $f_w(0)$ is a linear combination of $f_w(k)$ and that we secret shared the shares of $f_w(k)$ it now follows from our previous results (that shares for $\pm$ gates and multiplications by a constant can be computed locally), that now every player is able to compute his share of $f_w(0)$. More specifically, player i can now compute:

$$sh_{w,i} = \sum_{k=1}^{2t-1} sh_{w,k,i} \cdot \left(\prod_{l \neq k} \frac{i - x_l}{x_k - x_l} \right) = \sum_{k=1}^{2t-1} g_{w,k}(i) \cdot \left(\prod_{l \neq k} \frac{i - x_l}{x_k - x_l} \right)$$

Furthermore, we have:

$$\begin{aligned} f_w(0) &= \sum_{k=1}^{2t-1} g_{w,k}(0) \cdot \left(\prod_{l \neq k} \frac{x_l}{x_k - x_l} \right) = \sum_{k=1}^{2t-1} sh'_{w,k} \cdot \left(\prod_{l \neq k} \frac{x_l}{x_k - x_l} \right) \\ &= \sum_{k=1}^{2t-1} (f_u \cdot f_v)(k) \cdot \left(\prod_{l \neq k} \frac{x_l}{x_k - x_l} \right) = (f_u \cdot f_v)(0) = x_u \cdot x_v = x_w \end{aligned}$$

and since t players together can reconstruct $g_{w,k}(0)$ it follows that they can also reconstruct $f_w(0)$ as desired.

Security

Here, we only give a sketch of the security proof. The main intuition is as follows: At every step, each user "reveals" their value only through the t-out-of-n secret sharing scheme and thus, the adversary can learn nothing as long as he only sees the views of $t-1$ players. One crucial part in the construction is that $2t-1 \leq n \iff t < n/2$ must hold, since otherwise we can't use Lagrange Interpolation. Thus, we need what is called an "Honest Majority". Interestingly enough, it turns out that this is in fact optimal, i.e., as soon as we have a "Dishonest Majority", we cannot construct an information-theoretically secure MPC protocol. One consequence of this is that we cannot have information-theoretic security for 2PC.

Garbled Circuits

As we saw towards the end of the last section, computational security is the best we can hope for in the case of 2PC. Garbled Circuits are another mean to realize this, which turned out to have also a lot more application in Cryptography. On a high level they work as follows: A player would like to compute the value of a circuit on a given input, however, the circuit is only known to some other entity who does not wish to reveal anything about it to the player. Thus, he wants to give to the player a so called garbled circuit and a set of input labels which allow the player to compute the value of the circuit on his input, but in such a way that nothing else about the circuit is revealed.

More formally, we define the following two algorithm:

- $Garble(1^\lambda, C) \to \{L_{i,b}\}_{i\in[n],b\in\{0,1\}}, \hat{C}$
- $Eval(\hat{C}, \{L_{i,x_i}\}_{i\in[n]}) = C(x)$

where λ as usual is the security parameter, C is the circuit we wish to compute, $\{L_{i,b}\}$ the set of input labels, and $\hat{C}$ the garbled circuit. Furthermore, n is the length of x, i.e., the number of input wires of C. Security can be defined using a simulation security style definition like we saw last lecture, informally, we require that someone given $\hat{C}$ and $\{L_{i,x_i}\}_{i\in[n]}$ for some x can only learn $C(x)$ and nothing else about C, especially not $C(x')$ for some $x' \neq x$.

2PC using Garbled Circuits

In this section, we show how we can easily construct at 2PC protocol using garbled circuits, so for now we assume they exist and we can build them, we will show how to

do so in the next section. Suppose we have players A and B and they wish to compute $C(x)$. To put it in the words of our previous definition: Let C be the input of A, x be the input of B, $f_A(C, x) = \emptyset$, and $f_B(C, x) = C(x)$. The protocol is now as follows (here λ is the previously specified security of the protocol): A invokes $Garble(1^\lambda, C)$ to obtain $\{L_{i,b}\}_{i\in[n],b\in\{0,1\}}$ and $\hat{C}$. She now sends $\hat{C}$ to B, and wishes to send B all the input labels he needs to compute $C(x)$ but not more. B now wants to select these labels corresponding to x without revealing x. They can achieve this by engaging in an oblivious transfer (OT) for each i from 1 to n. After that, B can compute $C(x)$. Note, that during the OT A learns nothing about the bits of x and B learns nothing about the input labels he didn't choose. Thus, security of the protocol now follows from the security of Garbled Circuits.

Yao's Garbled Circuit

We now describe one construction of Garbled Circuits which is due to Yao, this requires a CPA-secure encryption scheme. For each wire w we choose 2 keys $k_{w,0}, k_{w,1} \xleftarrow{\$} \{0,1\}^\lambda$ which correspond to a value of 0 and 1 on the wire. For every input wire i, set $L_{i,b} = k_{i,b}$. Now, for every gate g with input wires u, v and output wire w, $b_0, b_1 \in \{0,1\}$ let d_{w,b_0b_1} be the following:

$$d_{w,b_0b_1} = Enc(k_{u,b_0}, Enc(k_{v,b_1}, k_{w,b_2}))$$

where $b_2 = g(b_0, b_1)$ is the output of gate (i.e., also the value of w) on input u, v. We compute d_{w,b_0b_1} for all four possible pairs of b_0b_1.

Let x_w be the value on wire w when computing $C(x)$. We now claim that given $\{L_{i,x_i}\}$ and $\{d_{w,b_0b_1}\}$ for all w we can compute k_{w,x_w} for all w. Note, that we have k_{i,x_i} for all input wires i, thus, we will try to compute the circuit gate by gate. Suppose g is a gate with input wires u, v and output wire w, we now do the following: For all four elements in $\{d_{w,b_0b_1}\}$ we try to decrypt twice to obtain k_{w,x_w}. Since we have k_{u,x_u} and k_{v,x_v} this can be done and one of the encryptions will be k_{w,x_w}. Note, that for this to work we need that decryption returns something unintelligible if we use the wrong key, one possible way to ensure this would be to pad the keys with a sufficiently long pattern (say a long string of 0s), making it unlikely that decryption with the wrong keys would yield exactly this pattern. Thus, we set $\hat{C} = \{\{d_{w,b_0b_1}\} \text{ for all wires } w\}, k_{out,1}$, where out denotes the output wire. Since $k_{out,1}$ was chosen at random it reveals nothing about the value of the output. However, it allows us to know the value of x_{out} whenever we have $k_{out,x_{out}}$ by simply comparing the two. It follows, that given $\hat{C}$ and $\{L_{i,x_i}\}_{i\in[n]}$ we can compute $C(x)$.

While this gives a valid *Eval* function it does not satisfy our security definition for two reasons:

First, if the $\{d_{w,b_0b_1}\}$ are always ordered in the same way (e.g., 00,01,10,11) then we can learn the value of the input wires of the gate by simply checking which decryption succeeded with respect to this ordering. Thus, we can reconstruct the whole circuit. To remedy this, we will choose a random ordering for $\{d_{w,b_0b_1}\}$ for each w and hence, whoever decrypts cannot do non-negligibly better than guessing what the input values were.

Second, while the scheme hides the actual gates of the circuit it does not hide its topology. Meaning, that we cannot learn the gates themselves but we can learn how they are connected to each other. This of course violates our security definition. To get around it, we introduce the notion of universal circuits. A universal circuit is a function (a circuit) U taking as arguments a circuit C' and an input x to C'. It then simulates the execution of C' on x and returns, $C'(x)$, i.e., $U(C', x) = C'(x)$. If we now want to garble a circuit C, what we instead to is garble a universal circuit $U(C, \cdot)$ with C hardcoded as a parameter. Note that the scheme described above still works in the same way, but since U has a fixed topology which is publicly known anyway, the *Garble* function know satisfies our security definition.

Outlook

Until now we have only considered the "Honest-but-Curious" setting for multiparty computation. In the next lecture we would like to drop the honesty assumption which will serve as an introduction to the topic of Zero-Knowledge Proofs.