COS 533: Advanced Cryptography — Princeton University
Lecture 6 (October 2, 2017)
Lecturer: Mark Zhandry — Scribe: James Bartusek

Notes for Lecture 6

1 Sender's Privacy and Simulation Security

Last time, we saw a protocol for oblivious transfer (OT), where Alice has two strings, x_0 and x_1, as her input, Bob has a bit b as his input, and Bob wants to know x_b. Sender's privacy stipulates that Bob 'learns nothing' about x_{1-b} from engaging in the protocol, and this was formalized as the inability for Bob to distinguish between any two distinct possibilities for x_{1-b} after having engaged in the protocol. However, consider the following two party computation task with this notion of sender's privacy in mind:

- Alice has sk for some signature scheme
- Bob has a message m
- They engage in a protocol at the end of which Bob outputs Sign(sk,m)

By sender's privacy, for any distinct sk_1, sk_2, Bob shouldn't be able to tell which one Alice used during execution of the protocol. However, there exist signature schemes which meet the definition of security we defined a few lectures back and which have the property that each signature on m could only possibly come from one sk. It seems that the indistiguishability notion of sender's privacy could not possibly be met if one of these signature schemes was used in the protocol. Therefore, we need a different way to capture the fact that Bob 'learns nothing' from engaging in the protocol. This motivates the notion of simulation security.

In a general two party computation task, Alice has input x, Bob has input y, and Bob wants to compute $f_B(x, y)$. Let $Trans(x,\, y)$ be the distribution over all possible sets of messages that are exchanged when evaluating $f_B(x, y)$ according to some protocol. We now say that the protocol satisfies sender's privacy if:

$$\exists \text{ PPT 'Simulator' } S \text{ s.t. } \forall x, y,\ t \leftarrow S(y, f_B(x, y)) \approx_c t \leftarrow Trans(x, y)$$

where $\approx_c$ means computationally indistinguishable. In other words, anything that Bob can compute after engaging in the protocol could have just been computed given only his input y and output $f_B(x, y)$. Now, going back to the signature scheme

example, if Bob could learn sk given only a message m and the signature Sign(sk,m) we would have a contradiction (assuming we instantiated the protocol with a secure signature scheme). Therefore, it is certainly possible in principal to design a protocol for this task that satisfies this new simulation security notion of sender's privacy.

2 Secret Sharing

Eventually, we will build general multi-party computation (MPC) from oblivious transfer and something called secret sharing. With secret sharing, there is a Dealer who holds a message m and n parties who each receive a share sh_i of m from the Dealer. The goal of n-out-of-n secret sharing is for all n parties to be able to reconstruct m but no set of n-1 parties to be able to. To formalize this, let $sh_i(m)$ be the share that party i receives of message m. Then, the above goal is satisfied if:

$$\forall S \subsetneq [n], \forall m_0, m_1, \{sh_i(m_0)\}_{i \in S} \approx \{sh_i(m_1)\}_{i \in S}$$

Now we describe a simple n-out-of-n scheme. Given message m, choose sh_1,sh_2,...,sh_n at random conditioned on the fact that $sh_1 \oplus ... \oplus sh_n = m$ (this is equivalent to choosing sh_1,...,sh_{n-1} at random and then setting $sh_n = m \oplus sh_1 \oplus ... \oplus sh_{n-1}$). It is easy to see that this satisfies the definition given above. Furthermore, this scheme has a useful linearity property, namely that $sh_i(m_0) \oplus sh_i(m_1) = sh_i(m_0 \oplus m_1)$. In other words, if all parties XOR their shares for two messages m_0 and m_1, the result will be a valid set of shares for the message $m_0 \oplus m_1$.

3 GMW Protocol for MPC

We now show a protocol for general functionality multi-party computation with an arbitrary number of parties. User i is given input $X_i = x_{i1}x_{i2}...x_{il}$. The function to compute f is given as a circuit C of $\oplus$ and $\wedge$ gates. As a setup, each user i computes shares sh_{ij} of X_i for $j = 1, ..., n$ and distributes sh_{ij} to user j. Now consider any input wire of C and let u and v be the two input bits and w be the correct output bit. Each user i has a share of the input bits, denoted sh_{ui} and sh_{uj}, and the goal will be to determine each user's share of the output bit w, denoted sh_{wi}. Now there are two possible gate types:

$\oplus$ gates: $sh_{wi} = sh_{ui} \oplus sh_{vi}$ (by the linearity of our secret sharing scheme)

$\wedge$ gates:

$$w = uv = (sh_{u1} \oplus ... \oplus sh_{un})(sh_{v1} \oplus ... \oplus sh_{vn}) =$$
$$[\bigoplus_i sh_{ui}sh_{vi}] \oplus [\bigoplus_{i<j}(sh_{ui}sh_{vj} \oplus sh_{vi}sh_{uj})] := Z \oplus [\bigoplus_{i<j} Y_{ij}]$$

Now the goal will be to construct shares of Z and of Y_{ij} for all i and j. Then each party can XOR their own shares of these values together to get their share of w. Z is particularly easy: just let $Z^{(i)} = sh_{ui}sh_{vi}$ where $Z^{(i)}$ is user i's share of the value Z. But now we need to come up with user k's share of $Y_{ij} = sh_{ui}sh_{vj} \oplus sh_{vi}sh_{uj}$. First of all, let $Y_{ij}^{(k)} = 0$ if $k \notin \{i, j\}$. Then have user i choose the value of $Y_{ij}^{(i)}$ at random from $\{0, 1\}$. Now, user i needs to communicate to user j what $Y_{ij}^{(j)}$ should be based on this choice but has no idea what the values of sh_{uj} and sh_{vj} are. Since there are only 4 possibilities for the values of those 2 bits, i can easily determine what $Y_{ij}^{(j)}$ should be in each case. The possibilties are shown in the following table.

sh_{uj}	sh_{vj}	$Y_{ij}^{(j)}$
0	0	$Y_{ij}^{(i)}$
0	1	$Y_{ij}^{(i)} \oplus sh_{ui}$
1	0	$Y_{ij}^{(i)} \oplus sh_{vi}$
1	1	$Y_{ij}^{(i)} \oplus sh_{ui} \oplus sh_{vi}$

Therefore, i and j can engage in a 1-out-of-4 OT protocol, where j requests the row in the above table coresponding to his true values of sh_{uj} and sh_{vj} and i sends the corresponding bit $Y_{ij}^{(j)}$, and where i learns nothing about the row that j requested and j does not learn the bit from any other row (given the value for two rows, j might be able to learn something about sh_{ui} or sh_{vi}). At the conclusion of this protocol, j has the correct value of $Y_{ij}^{(j)}$, and we are done. In this way, all parties can receive their shares of all the Y_{ij}s, and can then each separately compute their share of w.

Now we can imagine working up the circuit gate by gate, computing shares of the output of each gate in the way described above. Finally, each party will have their share of the output of the final gate, and combining these bits together, they will be able to compute the correct output of f.

Last time, we saw a 1-out-of-2 OT protocol, so all that is left to show is that a 1-out-of-4 OT protocol can be constructed using this 1-out-of-2 protocol. Denote the 4 possibilities (Alice's inputs) by x_1, x_2, x_3, x_4 and have Alice additionally choose two random bits a_1, a_2. Alice and Bob will then engage in a series of 3 1-out-of-2 OT protocols where Bob has the following choices:

1. x_1 or a_1
2. $x_2 \oplus a_1$ or $a_2 \oplus a_1$
3. $x_3 \oplus a_2$ or $x_4 \oplus a_2$

Now if Bob chooses x_1 in the first round, he will not have any information about a_1 and anything else he learns later on will just look random. Similar arguments can be shown for x_2, x_3, and x_4, so this protocol makes it impossible for Bob to learn more than 1 of the 4 xs, assuming the underlying 1-out-of-2 protocol is secure.