CS 276 – Cryptography Oct 22, 2014

Lecture 15: Secure multiparty computation I

Instructor: Sanjam Garg *Scribe: Shishir Agrawal*

1 Oblivious transfer

Rabin's oblivious transfer sets out to accomplish the following task. The sender has a bit $s \in \{0,1\}$. She places the bit in a box. Then the box reveals the bit to the receiver with probability 1/2, and reveals $\perp$ to the received with probability 1/2. The sender cannot know whether the receiver recieved s or $\perp$, and the received cannot have any information about s if they recieve $\perp$.

1-out-of-2 oblivious transfer sets out to accomplish the following related task. The sender has two bits $s_0, s_1 \in \{0,1\}$ and the receiver has a bit $c \in \{0,1\}$. The sender places the pair (s_0, s_1) into a box, and the receiver places c into the same box. The box then reveals s_c to the reciever, and reveals $\perp$ to the sender (in order to inform the sender that the reciever has placed his bit c into the box and has been shown s_c). The sender cannot know which of her bits the reciever recieved, and the reciever cannot know anything about s_{1-c}.

Lemma 1 *A system implementing 1-out-of-2 oblivious transfer can be used to implement Rabin's oblivious transfer.*

Proof. The sender has a bit s. She randomly samples a bit $b \in \{0,1\}$ and $r \in \{0,1\}$, and the reciever randomly samples a bit $c \in \{0,1\}$. If $b = 0$, the sender defines $s_0 = s$ and $s_1 = r$, and otherwise, if $b = 1$, she defines $s_0 = r$ and $s_1 = s$. She then places the pair (s_0, s_1) into the 1-out-of-2 oblivious transfer box. The reciever places his bit c into the same box, and then the box reveals s_c to him and $\perp$ to the sender. Notice that if $b = c$, then $s_c = s$, and otherwise $s_c = r$. Once $\perp$ is revealed to the sender, she sends b to the reciever. The recieiver checks whether or not $b = c$. If $b = c$, then he knows that the bit revealed to him was s. Otherwise, he knows that the bit revealed to him was the nonsense bit r and he regards it as $\perp$.

It is easy to see that this procedure satisfies the security requirements of Rabin's oblivious transfer protocol. Indeed, as we saw above, $s_c = s$ if and only if $b = c$, and since the sender knows b, we see that knowledge of whether or not the bit s_c recieved by the reciever is equal to s is equivalent to knowledge of c, and the security requirements of 1-out-of-2 oblivious transfer prevent the sender from knowing c. Also, if the reciever recieves r (or, equivalently, $\perp$), then knowledge of s is knowledge of the bit that was not revealed to him by the box, which is again prevented by the security requirements of 1-out-of-2 oblivious transfer. ■

Lemma 2 *A system implementing Rabin's oblivious transfer can be used to implement 1-out-of-2 oblivious transfer.*

Proof sketch. The sender has two bits $s_0, s_1 \in \{0,1\}$ and the reciever has a single bit c. The sender randomly samples $3n$ random bits $x_1, \ldots, x_{3n} \in \{0,1\}$. Each bit is placed into its own a Rabin oblivious transfer box. The ith box then reveals either x_i or else $\perp$ to the reciever. Let

$$S := \{i \in \{1, \ldots, 3n\} : \text{the reciever knows } x_i\}.$$

The reciever picks two sets $I_0, I_1 \subseteq \{1, \ldots, 3n\}$ such that $\#I_0 = \#I_1 = n$, $I_c \subseteq S$ and $I_{1-c} \subseteq \{1, \ldots, 3n\} \setminus S$. This is possible except with probability negligible in n. He then sends the pair (I_0, I_1) to the sender. The sender then computes the xor t_j of s_j together with x_i for all $i \in I_j$ for both $j \in \{0, 1\}$ and sends (t_0, t_1) to the reciever.

Notice that the reciever can uncover s_c from t_c since he knows x_i for all $i \in I_c$, but cannot uncover s_{1-c}. One can show that the security requirement of Rabin's oblivious transfer implies that this system satisfies the security requirement necessary for 1-out-of-2 oblivious transfer. ■

We will see below that length-preserving one-way trapdoor permutations can be used to realize 1-out-of-2 oblivious transfer.

2 Secure multiparty computation

Oblivious transfer is a special case of the following more general task, called *secure multiparty computation*. Suppose there are n people, and person i has access to some data x_i, and they are trying to compute some function of their inputs $f(x_1, \ldots, x_n)$. The goal is to do this securely: even if some parties are corrupted, no one should learn more than is strictly necessitated by the computation.

We have the following "ideal world" model for performing this task. Some of the people are honest, and each honest person i simply sends x_i to an angel. Some people are corrupted and are under control of a centralized adversary. The adversary chooses some input x_i' for each corrupted person i (where possibly $x_i' \neq x_i$) and that person then sends x_i' to the angel. The angel computes a function f of the values she recieves (for example, if only person 1 is honest, then the angel computes $f(x_1, x_2', x_3', \ldots, x_n')$) in order to obtain a tuple $(y_1, \ldots, y_n)$. She then sends person i back y_i. Honest people necessarily output y_i, and corrupted people can output anything. In any case, notice that at the end of this process, person i knows no more than the pair (x_i, y_i), so this model formally realizes our intuition earlier that no person has learnt more than strictly necessary.

In the real world, there is no angel for performing these computations. In any case, we can define the security of a secure multiparty computation protocol by comparison to the ideal world protocol described above. There are a lot of variants of precise definitions depending on what kind of capabilities we want to allow from the adversary. They can be computationally bounded (which means that they must run in probabilistic polynomial time) or unbounded, they can be semihonest (which means that they follow the protocol exactly but attempt to learn information that is supposed to be kept private) or malicious (which means that they might deviate arbitrarily from the protocol)...

For modelling malicious adversaries, it turns out that the above "ideal world" model is a bit too strong, so we weaken it as follows. Each person sends either x_i or some other value x_i' (again chosen by the adversary, as above) to the angel, depending on whether they are honest or corrupted, respectively. The angel computes f applied to the values she recieves to obtain a value $y := (y_1, \ldots, y_n)$. She then sends y to the attacker, who gets to decide whether or not honest parties will recieve their response from the angel. The angel obliges. Each honest person i then outputs y_i if they recieve y_i from the angel and $\perp$ otherwise, and corrupted parties output whatever the adversary tells them to.

For example, a protocol Π is secure against computationally bounded adversaries if for all tuples $A = (A_1, \ldots, A_n)$ of probabilistic polynomial time machines, there exists a tuple $S = (S_1, \ldots, S_n)$ of "simulating" probabilistic polynomial time machines such that for all tuples of bit strings $(x_1, \ldots, x_n)$ such that $|x_1| = \cdots = |x_n|$, we have

$$\text{Real}_{\Pi,A}(x_1, \ldots x_n) \stackrel{c}{\simeq} \text{Ideal}_{f,S}(x_1, \ldots, x_n)$$

where the left-hand side denotes the output distribution induced by Π running with A representing all of the people involved, and the right-hand side denotes the output distribution induced by running the ideal protocol with S representing all of the people involved. The ideal protocol is either the original one described for semi-honest adversaries, or the modified one described in the previous paragraph for malicious ones.

Theorem 3 *The following protocol realizes 1-out-of-2 oblivious transfer in presence of computationally bounded and semihonest adversaries.*

1. *The sender, who has two bits s_0 and s_1, samples a random length-preserving one-way trapdoor permutation (f, f^{-1}) and sends f to the reciever.*
2. *The reciever, who has a bit c, randomly samples a n bit strings $x_c \in \{0,1\}^n$ and computes $y_c = f(x_c)$. He then samples a second random n bit string $y_{1-c} \in \{0,1\}^n$, and then sends (y_0, y_1) to the sender.*
3. *The sender computes $x_0 := f^{-1}(y_0)$ and $x_1 := f^{-1}(y_1)$. If b is a hard-core bit for f, she then computes $b_0 := b(x_0) \oplus s_0$ and $b_1 := b(x_1) \oplus s_1$, and she then sends the pair (b_0, b_1) to the reciever.*
4. *The reciever knows c and x_c, and can therefore compute $s_c = b_c \oplus b(x_c)$.*