CS 276 – Cryptography Oct 27, 2014

Lecture 16: Secure multi-party computation (GMW Protocol + Malicious Model)

Instructor: Sanjam Garg *Scribe: Wenting Zheng*

1 Secure two-party computation

The two-party secure computation protocol ensures that a joint computation is carried out between two parties without either party learning anything beyond the final value of the evaluation. Specifically, we do not want any party to learn anything beyond its own input into the joint computation. The first part of the lecture notes describes a specific protocol for performing a joint computation with two parties' inputs using boolean circuits.

Problem setup

There are two parties, P_1 and P_2. P_1's input is x, P_2's input is y. We assume that both inputs are of the same length: $|x| = n$, $|y| = n$. P_1 and P_2 evaluate a boolean circuit denoted by C. This boolean circuit can have different types of gates, such as XOR, AND, NAND, etc. Each gate has either one or two inputs, and one output. We construct the gate such that it evaluates one bit at a time. We denote the input to the circuit to be x and y, where x is the input from P_1, and y is the input from P_2.

We assume that the parties are honest but curious, in that they perform the actions correctly, but attempt to learn about other parties' inputs.

Intuition

Each party splits his/her input into two parts, then distribute one part to the other party, while keeping one part to him/herself. The parties then go through each gate in the boolean circuit together, computing results on their shares together. After every gate computation, each party should still keep a part for itself. Put it another way, each party should have one share per wire, for a total of two shares per wire.

Protocol (GMW87)

Initialization

$x \to \{x_1, x_2, ..., x_n\}$, where x_i denotes the i^{th} bit of x. Similarly, $y \to \{y_1, y_2, ..., y_n\}$. We denote W as the set of wires in the circuit. For every wire w, its value is denoted by p_w. $p_w = a_w \oplus b_w$. P_1 takes the a_w shares, while P_2 takes all of the b_w shares. Both parties start by splitting their inputs into a and b shares. For example, P_1 splits x in the following way:

$\forall w \in \{w_1, w_2, ..., w_n\}$
Sample $a^1_{w_i} \leftarrow \{0, 1\}$
$b^1_{w_i} = x_i \oplus a^1_{w_i}$

Similarly, P_2 splits y into $a^2_{w_i}$ and $b^2_{w_i}$, where $i \in \{1, 2, ..., n\}$. After the splits are done, P_1 keeps all of the a_w shares, while P_2 keeps the b_w shares.

Evaluation

Here we will give three example gate evaluations. We assume that P_1 has two shares, $a^1_{w_i}$ and $a^2_{w_i}$, while P_2 has two shares $b^1_{w_i}$ and $b^2_{w_i}$. $x_i = a^1_{w_i} \oplus b^1_{w_i}$, and $y_i = b^1_{w_i} \oplus b^2_{w_i}$. Given a gate G, we wish to evaluate $G(x_i, y_i)$ without P_1 learning about the value of y_i, or P_2 learning the value of x_i.

XOR gate

To evaluate $x_i \oplus y_i$, we can have the two parties evaluate their shares separately, since $x_i \oplus y_i = (a^1_{w_i} \oplus b^1_{w_i}) \oplus (a^2_{w_i} \oplus b^2_{w_i})$. Therefore, P_1 will evaluate $a^1_{w_i} \oplus a^2_{w_i}$, and P_2 will evaluate $b^1_{w_i} \oplus b^2_{w_i}$.

AND gate

In this situation, we wish to evaluate $x_i \wedge y_i = (a^1_{w_i} \oplus b^1_{w_i}) \wedge (a^2_{w_i} \oplus b^2_{w_i})$. We would still like the two parties to finish the evaluation with one share each, but doing so without revealing each other's initial share. In order to realize this, we use 1-4 oblivious transfer (OT) protocol.

We denote $T = x_i \wedge y_i$. P_1 randomly samples $\sigma \leftarrow \{0,1\}$. It then constructs the following table:

b_1	b_2	$T = (a^1_{w_i} \oplus b_1) \wedge (a^2_{w_i} \oplus b_2))$	s
0	0	α_0	$s_0 = \sigma \oplus \alpha_0$
0	1	α_1	$s_1 = \sigma \oplus \alpha_1$
1	0	α_2	$s_2 = \sigma \oplus \alpha_2$
1	1	α_3	$s_3 = \sigma \oplus \alpha_3$

$P1$ then uses the 1-4 OT protocol to transfer over (s_0, s_1, s_2, s_3). P_2 then uses its shares $(b^1_{w_i}, b^2_{w_i})$ to retrieve the correct result. P_2 keeps the result, while P_1 keeps σ. The details of the 1-4 OT protocol will be explored in the next section.

NOT gate

For the NOT gate, we wish to evaluate the inverse of a particular input w_i. Since for each wire, each party P_1 and P_2 keeps one share. We denote a_{w_i} to be the share kept by P_1, and b_{w_i} to be the share kept by P_2. Thus, we have $w_i = a_{w_i} \oplus b_{w_i}$. The result of the gate is $\neg w_i$.

In order to achieve this, we simply denote the new share for P_1 to be $a'_{w_i} = \neg a_{w_i}$ (or $1 - a_{w_i} \mod 2$), and the new share for P_2 to be $b'_{w_i} = \neg b_{w_i}$ (or $1 - b_{w_i} \mod 2$). This is because $\neg w_i = (\neg a_{w_i}) \oplus (\neg b_{w_i})$.

1 in 4 OT Protocol

In this section, we will describe a construction for a secure 1 in 4 oblivious transfer protocol.

The aim of the protocol is to transfer values between two parties securely. We would like to guarantee that the sender does not learn which value the receiver picked, and the receiver does not know the values that it did *not* pick. In particular, 1-4 OT protocol has the sender send over four values, and the receiver is only supposed to receive on of these values without knowing anything about the other three values.

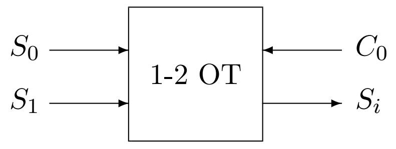
Thus, we have two parties, sender S and receiver R. Suppose S transfers four messages denoted by $(m_{00}, m_{01}, m_{10}, m_{11})$, and R uses two bits (C_0, C_1) to select one of the values from S.

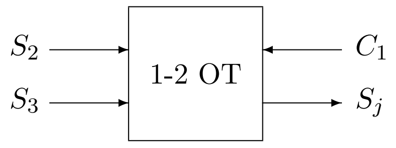
S first calculates the following:

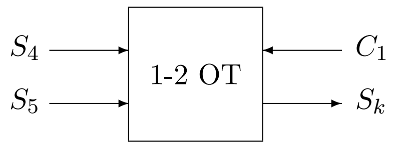
1. Sample $S_i \leftarrow \{0, 1\}$ for $i \in \{0, 1, ...5\}$.
2. Calculate the following:

$$
\begin{aligned}
\alpha_0 &= S_0 \oplus S_2 \oplus m_0 \\
\alpha_1 &= S_0 \oplus S_3 \oplus m_1 \\
\alpha_2 &= S_1 \oplus S_4 \oplus m_2 \\
\alpha_3 &= S_1 \oplus S_5 \oplus m_3
\end{aligned}
$$

S and b se three 1-2 OT protocols







Using (C_0, C_1), R is able to decode one of the four messages sent by S.

2 Malicious attacker intead of semi-honest attacker

The assumption we had before consisted of a semi-honest attacker instead of a malicious attacker. This means that the attacker does not have to follow the protocol, and may instead alter the original protocol. The main idea here is that we can convert a protocol aimed at semi-honest attackers into one that will work with malicious attackers.
At the beginning of the protocol, we have each party commit its inputs:
Given a commitment protocol *com*, Party 1 produces

$$\begin{aligned} c_1 &= com(x_1; w_1) \\ d_1 &= com(r_1; \phi_1) \end{aligned}$$

Party 2 produces

$$\begin{aligned} c_2 &= com(x_2; w_2) \\ d_2 &= com(r_2; \phi_2) \end{aligned}$$

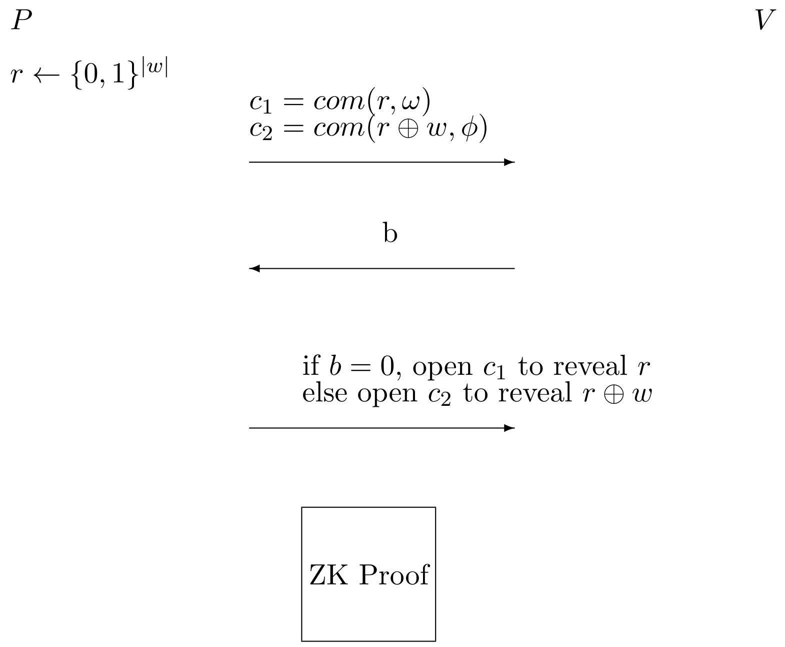
We have the following guarantee: $\exists x_i, r_i, w_i, d_i$ such that $c_i = com(x_i; w_i) \wedge d_i = com(r_i; \phi_i) \wedge t = \pi(i,$ transcript $, x_i, r_i)$, where transcript is the set of messages sent in the protocol so far.
Here we have a potential problem. Since both parties are choosing their own random coins, we have to be able to enforce that the coins are *indeed* random. We can solve this by using the following protocol:

$$\begin{array}{ccc} \xrightarrow{d_1 = com(s_1, \phi_1)} & \qquad\qquad & \xleftarrow{d_2 = com(s_2, \phi_2)} \\ \xrightarrow{s_2^{'}} & \qquad\qquad & \xleftarrow{s_1^{'}} \end{array}$$

We calculate $r_1 = s_1 \oplus s_1^{'}$, and $r_2 = s_2 \oplus s_2^{'}$. As long as one party is able to choose randomly, both parties wil be enforced to choose randomly.
Furthermore, during the first commitment phase, we want to make sure that the committing party actually knows the value that is being committed to. Thus, we also attach along with the commitments a zero knowlwedge proof of knowledge (ZK-PoK) to prove that the committing party is committing a valid value.

2.1 Zero knowledge proof of knowledge (ZK-PoK)

Definition: Zero knowlwedge proof of knowledge (ZK-PoK) is a zero knowledge proof system with the property proof of knowledge:
$\exists$ a PPT E (knowledge extractor) such that $\forall x \in \{0,1\}^n$ and $\forall P^*$ (possibly unbounded) for which $Pr[Out_v(P^*(x) \leftarrow v(x)) = 1] \geq \epsilon(n)$, then
Here we have L be the language, R be the relation, and $R(x)$ is the set such that $\forall w \in R(x)$, $(x, w) \in R$.
Given a zero knowledge proof system, we can construct a ZK-PoK system for statement x as follows:


P
V
$r \leftarrow \{0,1\}^{|w|}$
$c_1 = com(r, \omega)$
$c_2 = com(r \oplus w, \phi)$
b
if $b = 0$, open c_1 to reveal r
else open c_2 to reveal $r \oplus w$
ZK Proof

The proof system proves that $\exists r, w, \omega, \phi$ such that $(x, w) \in R$ and $c_1 = com(r; \omega)$, $c_2 = com(r \oplus w; \phi)$.