University of Illinois, Urbana Champaign
CS/ECE 498AC3/4 Applied Cryptography

*Instructor:* Dakshita Khurana
*Scribe:* Aaron Gros
*Date:* November 10, 2020

**LECTURE**
**22**

# Two Party Computation 2: Garbled Circuits

In this lecture we will begin by reviewing and expanding on secret sharing, and introducing garbled circuits. We've been discussing how two parties can complete computations without sharing each other's data. In the previous lecture we discussed secret sharing as a method of sharing enough information to do the computation but not reveal each other's data or any of the results of layers before the output for nested circuits. In today's lecture we will review how secret sharing works and cover an issue with our use of oblivious transfer(OT) in the way we implemented secret sharing. We will go through how to fix this issue by adding randomness into the input of the OT that will later be undone. Finally we will propose a method known as garbled circuits for two-party computation where only one party knows what the computation is. We will begin with a simple example to gain intuition and then expand to show how you could build a more complex and useful garbled circuit.

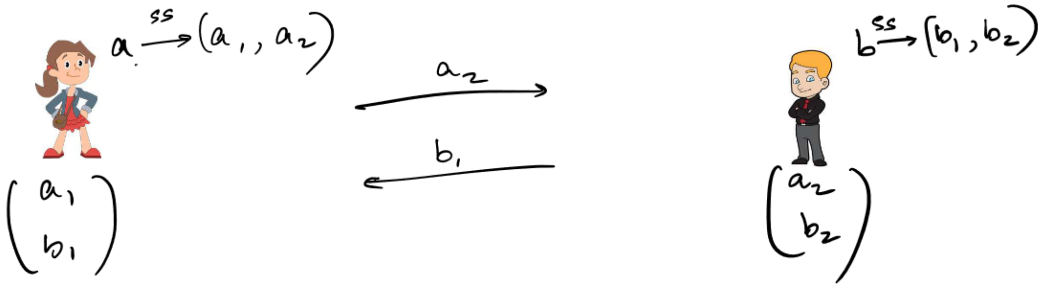

$a \xrightarrow{ss} (a_1, a_2)$
$a_2$
$b_1$
$b \xrightarrow{ss} (b_1, b_2)$
$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}$
$\begin{pmatrix} a_2 \\ b_2 \end{pmatrix}$

FIGURE 22.1: Basic interaction for secret sharing.

## 22.1 Background

In the previous lecture we covered secret sharing and computation. This method is depicted in Figure 22.1. Both parties(in this case Alice and Bob) compute values that when combined form their data. In this case Alice computes $a_0, a_1$ such that $a_0 \oplus a_1 = a$ by computing random $a_0$ and finding $a_1$ such that the xor is satisfied. Bob does the same with his data to obtain $b_0, b_1$ such that $b_0 \oplus b_1 = b$. The separation is not limited to two values but in this example it's two for simplicity's sake. Once these are computes, Alice shares $a_1$ with Bob and Bob shares $b_0$ with Alice. From there Alice can compute

$$(a_0 \wedge b_0) \oplus (a_1 \wedge b_0)$$

and Bob can compute

$$(a_1 \wedge b_1)$$

We Then use an OT as depicted by Figure 22.2 where Alice inputs $a_0 \wedge 0$ and $a_0 \wedge 1$ and Bob inputs $b_1$ to receive $a_0 \wedge b_1$. Then Alice sends Bob $(a_0 \wedge b_0) \oplus (a_1 \wedge b_0)$ and Bob sends Alice $(a_0 \wedge b_1) \oplus ((a_1 \wedge b_1))$. Now both of them can compute

$$\begin{aligned}(a_0 \wedge b_0) \oplus (a_1 \wedge b_0) \oplus (a_0 \wedge b_1) \oplus ((a_1 \wedge b_1)) &= (a_0 \oplus a_1) \wedge (b_0 \oplus b_1) \\ &= a \wedge b\end{aligned}$$

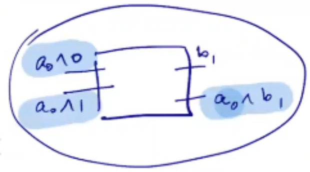


FIGURE 22.2: Original OT for AND secret sharing.

## 22.2 Issue with our protocol

In this section we will cover an issue with the implementation of secret sharing we previously covered and will provide a way to fix this issue. When the OT depicted in Figure 22.2 is used for secret sharing, Bob receives $a_0 \wedge b_1$. This is a problem because he also has $a_1 \wedge b_1$ so he can compute

$$\begin{aligned}a_0 \wedge b_1 \oplus a_1 \wedge b_1 &= (a_0 \oplus a_1) \wedge b_1 \\ &= a \wedge b_1\end{aligned}$$

If Bob chooses $b_1 = 1$ he can find out what a is which breaks out requirement that neither Alice nor Bob be able to find out what the others data is beyond what can be recovered from the final output.

A solution to this problem would be to slightly modify the inputs of the OT in a reversible way. To do this, Alice will need to generate a random value r and instead we will use the OT depicted in Figure 22.3

Alice will input $(a_0 \wedge 0) \oplus r$ and $(a_0 \wedge 1) \oplus r$ and Bob inputs $b_1$ to receive $(a_0 \wedge b_1) \oplus r$. Then Alice sends Bob $(a_0 \wedge b_0) \oplus (a_1 \wedge b_0) \oplus r$ and Bob sends Alice $(a_0 \wedge b_1) \oplus r \oplus ((a_1 \wedge b_1))$. Now both of them can compute

$$\begin{aligned}(a_0 \wedge b_0) \oplus (a_1 \wedge b_0) \oplus r \oplus (a_0 \wedge b_1) \oplus r \oplus ((a_1 \wedge b_1)) &= (a_0 \oplus a_1) \wedge (b_0 \oplus b_1) \\ &= a \wedge b\end{aligned}$$

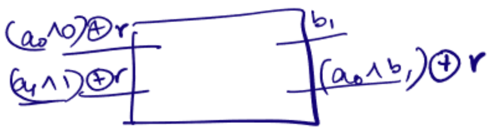

$(a_0 \wedge 0) \oplus r$
$(a_1 \wedge 1) \oplus r$
$b_1$
$(a_0 \wedge b_1) \oplus r$

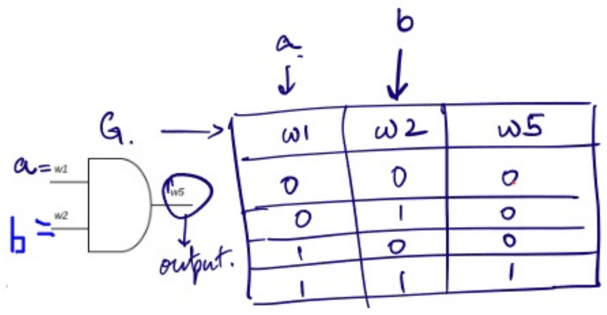
FIGURE 22.3: New OT for AND secret sharing.

With this method, the XOR with r cancels out in the final computation but Bob isn't able to recover a since the output of the OT is now a random value.

## 22.3 Garbled Circuits

Imagine a situation where Alice has a circuit in mind that she doesn't want to share with Bob but she still wants Bob to be able to compute the circuit with his data. However, neither Alice or Bob wants to share their data with each other. To deal with such a situation we propose a protocol called a Garbled Circuit.

To describe the garbled circuit protocol, we first begin with a circuit that Alice has in mind. In this case we will start with a single logic gate depicted in Figure 22.4 to gain intuition. We turn this logic gate circuit into a logic table again as depicted in Figure 22.4. Now we will make a new table called the garbled table depicted by Table 22.1 which we will send to bob. Briefly, for intuition, our objective will be to represent the logic table by a single column table which will contain each output double encrypted with specific keys relating to the inputs so that when Bob receives the garbled table he can only decrypt one row.



| w1 | w2 | w5 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

FIGURE 22.4: Alice's Circuit Table.

| |
|---|
| $E_{k_{w1,0}}(E_{k_{w2,0}}(K_{w5,0}))$ |
| $E_{k_{w1,0}}(E_{k_{w2,1}}(K_{w5,0}))$ |
| $E_{k_{w1,1}}(E_{k_{w2,0}}(K_{w5,0}))$ |
| $E_{k_{w1,1}}(E_{k_{w2,1}}(K_{w5,1}))$ |

TABLE 22.1: Alice's Garbled Table

To create this table, Alice will generate two keys for each input and output. We will label these $K_{w1,0}, K_{w1,1}, K_{w2,0}, K_{w2,1}, K_{w5,0}, K_{w5,1}$ for inputs w1, w2 and output w5, respectively. $K_{w1,0}$ corresponds to w1 input 0 and $K_{w1,1}$ corresponds to w1 input 1 and so on for the w2 and w5 keys. To perform the encryption, we set the w5 keys as the message to encrypt and first encrypt with the w2 keys and then the w1 keys following the logic table as a guide. The order between w1 and w2 is arbitrary but should be kept the same throughout the entire process. Beginning with the topmost row of the logic gate, we will encrypt $K_{w5,0}$ with $K_{w2,0}$ and encrypt the result with $K_{w1,0}$. This process will the repeated for the other rows following Table 22.1.

Once the garbled table is complete, Alice sends Bob the garbled table, a mapping for which output key corresponds to which output($K_{w5,0} \rightarrow 0, K_{w5,1} \rightarrow 1$), and the key corresponding with Alice's input(i.e. $K_{w1,a}$). With Alice's key, Bob will be able to remove one layer of encryption of two of the rows in the garbled table. To receive the key corresponding to Bob's input we will use an OT where Alice inputs both of Bob's keys $K_{w2,0}, K_{w2,1}$, and Bob inputs his data b. This allows Bob to receive $K_{w2,b}$ without Alice finding out what his b is. Finally, Bob will use $K_{w2,b}$ to decrypt only the row corresponding to both Alice's and Bob's input. Once he receives the output key, Bob can use the mapping to find out what the output was.

Three quick notes:

- We set the keys corresponding to the output w5 be the output of the decryption instead of the output of the logic gate to generalize to when several logic gates are chained together and decrypting the output of the logic gate would be leaking information.
- Similarly, we send all four row values instead of just the two corresponding with Alice's input to generalize for when multiple logic gates are chained together or for when Bob has both inputs.
- The type of encryption used here is authenticated encryption so both Bob and Alice will know if Bob used the right keys for the right encryptions.

An important issue to address is that if the garbled table is passed as it is, Bob could know which row corresponds to which input which would leak information. Therefore, Alice has to randomly permute the rows before sending the table to Bob.

## 22.4 Multi Layered Garbled Circuits

In the last section we covered how garbled circuits look like for a circuit with just a single logic gate, in this section we will cover how to use garbled circuits with logic gates chained together.

In this example, Alice's circuit is depicted by Figure 22.5. Similarly to single logic gate example we will create a garbled table, however, in this case we will create one for each logic gate. To do so we will begin by generating two keys for each input of every logic gate, and two keys for the final output. Then we will generate the three garbled tables as shown in Table 22.2.

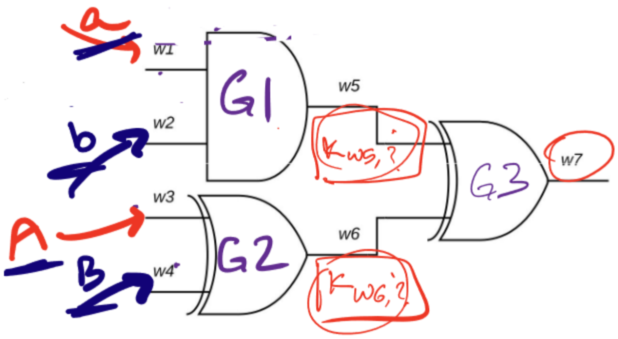


FIGURE 22.5: Alice's multi layered circuit.

| Garbled Table G1 | Garbled Table G2 | Garbled Table G3 |
|---|---|---|
| $E_{k_{w1,0}}(E_{k_{w2,0}}(K_{w5,0}))$ | $E_{k_{w3,0}}(E_{k_{w4,0}}(K_{w6,0}))$ | $E_{k_{w5,0}}(E_{k_{w6,0}}(K_{w7,0}))$ |
| $E_{k_{w1,0}}(E_{k_{w2,1}}(K_{w5,0}))$ | $E_{k_{w3,0}}(E_{k_{w4,1}}(K_{w6,1}))$ | $E_{k_{w5,0}}(E_{k_{w6,1}}(K_{w7,1}))$ |
| $E_{k_{w1,1}}(E_{k_{w2,0}}(K_{w5,0}))$ | $E_{k_{w3,1}}(E_{k_{w4,0}}(K_{w6,1}))$ | $E_{k_{w5,1}}(E_{k_{w6,0}}(K_{w7,1}))$ |
| $E_{k_{w1,1}}(E_{k_{w2,1}}(K_{w5,1}))$ | $E_{k_{w3,1}}(E_{k_{w4,1}}(K_{w6,0}))$ | $E_{k_{w5,1}}(E_{k_{w6,1}}(K_{w7,0}))$ |

TABLE 22.2: Alice's multi layered circuit garbled tables

Note that the keys for w5 and w6 are used as the message encrypted for circuits G1 and G2 and then used as keys for G3.

Once the garbled tables are created we can send them to Bob, along with the mapping of the final output($K_{w7,0} \to 0, K_{w7,1} \to 1$), and they keys for w1 and w3 corresponding to Alice's data(i.e. $K_{w1,a}$ and $K_{w3,A}$). Then we will use two OTs as described in the single logic gate example but this time to share the key for w2 and w4 corresponding to Bob's data.

Now, Bob has the keys to decrypt one row of G1 and one row of G2. The result from these decryptions can then be used to decrypt a row of G3 and get the final output of the circuit using the map.

Note that Bob only knows the key for w5 and w6 but not the actual value the keys correspond to. This is imortant because the garbled circuit isn't leaking information.

## Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov.