# 1 Lecture 14: Garbled Circuits

The basic idea of garbled circuits is to do multi-party computation with a constant number of communication rounds. As the digital circuits get larger, evaluation via explicitly communicated secret-sharing for every wire can become very expensive. As it will turn out, we can do multiple computations, even ones with several layers of depth dependent on one another, all in one round of computation.

A basic level of security that garbled circuits should pass is, given a function $f$ and two inputs $x_0, x_1$, along with the garbled version of the function, and a garbled version of one input, we should not be able to tell which input was garbled and given back to us. (i.e., no negligible advantage over guessing it w/ probability 1/2).

For garbled circuits, we use the fact that NOT and AND form a universal set of gates (as you can create NOR, and thus OR, by DeMorgan's Laws). This is in contrast to how we used $\otimes$ and $\oplus$ for the original MPC digital circuits.

At a high level, one player will create a "garbled circuit" and garble the inputs, and send this info to the other player, who will then compute a garbled output, which will be sent back to the other player to be decrypted, to give the true final output. We use the following steps:

1. The computation is converted into a circuit of AND's and NOT's
2. The circuit is garbled, and the "garbler" garbles their own inputs as well
3. The garbled circuit and one party's garbled inputs are sent to the other party
4. The other party's inputs are garbled via 1-2-OT.
    (a) The garbler is the sender, and the evaluator is the receiver. The garbler will send an encryption of a 0 and an encryption of a 1, and the receiver chooses the bit corresponding to their true input.
5. The "evaluator" then evaluates the garbled circuit
6. The garbler decrypts this garbled output, so both parties can know the true output of the circuit.

Essentially, a garbled circuit simply consists of a set of tables (which represent the gates), and a set of pairs of strings (which represent possible values (0,1) for each wire). The evaluator of the circuit uses the tables one-by-one, plugging in past strings to figure out future strings, and eventually ending up with the final garbled output. So, along the way, the evaluator should learn which key is used for each wire, but should not learn the significance of the key (i.e., whether it is a 0 or a 1). However, note that if we just use keys in the tables, there is an obvious problem. For example, if we have a garbled AND gate, three of the outputs will be the same, and the evaluator will be able to tell that these represent 0's.

So, we have to additionally encrypt each output from the table. A couple options are given for this. In the lecture, we propose using the inputs corresponding to the output to encrypt that output, for example

| $A$ | $B$ | $A\&B$ | $\rightarrow$ | Encrypted $A\&B$ |
|---|---|---|---|---|
| $A_0$ | $B_0$ | $C_0$ | $\rightarrow$ | $E_{A_0}(E_{B_0}(C_0))$ |
| $A_0$ | $B_1$ | $C_0$ | $\rightarrow$ | $E_{A_1}(E_{B_0}(C_0))$ |
| $A_1$ | $B_0$ | $C_0$ | $\rightarrow$ | $E_{A_0}(E_{B_1}(C_0))$ |
| $A_1$ | $B_1$ | $C_1$ | $\rightarrow$ | $E_{A_1}(E_{B_1}(C_1))$ |

Of course, if we always keep the rows in the same order, this gives away some information, so we will shuffle the rows around before sending the table.

In Rafi's notes from 282A, we see a different scheme which involves applying a PRG to the inputs and masking each output with segments of the pseudo-random strings from its corresponding inputs (i.e., $C_0 \rightarrow G(A_0)[0:n] \oplus G(B_0)[3n:4n] \oplus C_0$). Either way, the general idea is the same, and by some method, the evaluator will not know what it is they are evaluating.

It is intuitive that this should work for multi-party computation, and neither party should learn anything besides what is implied by the output.

## 1.1 In Practice

MPC is used widely in practice, and Intel's AES accelerator allows garbled circuits to be computed very quickly (i.e., billions of gates per second). We see it used between companies like Google and Visa, who want to share data to make joint calculations, but have users with privacy concerns whose data cannot be explicitly shared. We also see it used in studies to figure out gender wage gaps, where individual companies do not want to share their own employees' wages. It is also used to determine whether satellite trajectories will intersect, between governments who do not want to reveal important information about their own assets. Finally, it has also been proposed to be used for a "student right to know before you go" bill, in which students would know what starting salary they are likely to have for a degree. This would require pooling graduation poll data from many universities, which would likely have to be done by MPC to keep individual school and student data private.

## 1.2 Dealing with Malicious Players

Research on dealing with malicious players has been done more recently.

The weakest notion of security for garbled circuits is indistinguishability, where, given a function $f$ and two inputs $x_1, x_2$ such that $f(x_1) = f(x_2)$, we should not be able to tell apart an encryption of $E(x_1)$ and $E(x_2)$. This ensures that the evaluator cannot determine what output is actually being computed for each wire.

A stronger notion of security is simulation. Like zero-knowledge, we are looking for some way for each player to simulate the other player's actions, without that player being present. This should allow the protocol to work if one player is malicious.

A separate idea is adaptive corruption, where a player may start by behaving honestly, but then may become corrupt later on in the protocol. We can deal with this by using a partial simulator, where we use the true transcript up to a certain point, then simulate the rest of the messages.

Finally, we have the strongest idea, which is universal composability. This asserts that MPC should remain secure even when it is being used as a smaller part of a larger protocol. Clearly, this is useful, but we would have to deal with players becoming malicious before or after the MPC is done, which makes this very hard to prove.

We can deal with a malicious garbler with a zero-knowledge proof that: (1) the garbler's inputs are consistent, and (2) the gates have been garbled correctly (i.e., not changing the circuit).

We enforce that the evaluator behaves by making them request a digital signature for each output they open, forcing them to only open the outputs they absolutely need, since they would need to forge a signature in order to gain more information.

## 1.3 Definitions of Security

We define a basic game for non-adaptive security, where given an input and a function, a garbler must be able to produce a garbled version of the input and function, respectively. In order to show that our garbling scheme is secure, we want to ensure that if a simulator is given the circuit and the desired output but does not know how to actually garble anything (i.e., does not create an actual garbled circuit, but instead some fake circuit that gives the desired output), the adversary should not be able to tell the difference.

The adaptive version is one where the garbler must provide the garbled circuit before the input is given, which makes this much harder for a simulator, since they cannot simply create a fake circuit that outputs a single desired output, since they will not know what the output should be when making the circuit. However, we do not cover this security definition during this class.

The Linkdell-Pinkas proof of non-adapative (aka selective) security for garbled circuits essentially comes down to a hybrid argument, where the hybrids combine real garbling (as described at the beginning of this lecture) and simulated garbling (described above, with a "fake" circuit that just outputs the desired value every time). In each hybrid, we break an additional gate so that no matter what its inputs are, it will always output the desired result, no matter what the inputs are. Since the evaluator doesn't know the encryption keys for each wire, they can't tell the difference unless somehow the output changes. But since the broken hybrids keep the output the same, there is no way for the evaluator to tell the difference. Thus, we can't tell the difference between any two hybrids (where only one additional gate is broken), so we can't tell the difference between an actual garbling and a simulated garbling, so we pass the selective security as desired.

### 1.4 Beaver's OT Extension

We know that directly constructing 1-2-OT is expensive, and that garbled circuits tend to be large, but they are cheap to send. So, Beaver came up with an idea that we can use to convert $k$ OT's into polynomially many OT's by using garbled circuits.

The basic building block here is his theorem (Theorem 1) from the paper:

D. Beaver. Correlated Pseudorandomness and the Complexity of Private Computations. In the 28th STOC, pp. 479–488 (1996)

Essentially, if a one-way function exists, then there is a protocol to compute polynomially many oblivious transfers given a constant number. So, we can implement a garbled circuit to perform this protocol via the one-way function, and thus we have that we can use garbled circuits to generate more 1-2-OT's.