Garbled Circuit Optimizations

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2025

Last time, we completed an instantiation of a 2PC preprocessing functionality for malicious Boolean circuits. We used that preprocessing functionality to instantiate an *interactive* MPC protocol, where the parties expend communication rounds proportional to the multiplicative depth of their desired computation.

Recall that in the semi-honest setting, we were able to achieve MPC protocols that run in only a small constant number of rounds by leveraging the *garbled circuit* technique. Our goal over the next two lectures is to achieve maliciously-secure 2PC protocols that run in a constant number of rounds. This will require us to re-examine the garbled circuit technique.

In particular, today we will not *yet* discuss malicious security. Instead, we will explore *optimizations* to the standard notion of semi-honest garbling. These optimizations are interesting in their own right, and are the result of a significant effort by the research community. In addition, the optimizations we will see today will make it easy to understand how to achieve the maliciously-secure construction we will see next time.

In particular, we will introduce powerful optimizations to the garbled circuit technique: the Free XOR technique and the half gates technique [KS08, ZRE15].

Garbled Circuit Review

Recall that in garbled circuits, we assign our two compute parties asymmetric roles. The so-called *garbler* prepares an encoding of the circuit, and conveys encodings of circuit inputs via OT. The so-called evaluator uses those encodings to evaluate the circuit "under encryption", obliviously obtaining an encoding of the output, which the parties jointly decrypt.

The main idea was to assign to each wire x in the circuit two keys k_x^0, k_x^1. One key encodes a logical zero, the other encodes a logical one. From here, the idea was to encrypt the logical rows of the gate of each truth table. In particular, we used all four combinations of gate input keys to encrypt corresponding output keys. For example, an AND gate could be encrypted as follows:

$$
\begin{aligned}
&\texttt{GarbleAND}((K_x^0, K_x^1, K_y^0, K_y^1, K_z^0, K_z^1)):\\
&\quad (k_x^0, c) \leftarrow K_x^0\\
&\quad (k_x^1, \cdot) \leftarrow K_x^1\\
&\quad (k_y^0, d) \leftarrow K_y^0\\
&\quad (k_y^1, \cdot) \leftarrow K_y^1\\
&\quad r_{00} \leftarrow \texttt{Enc}(k_x^c, \texttt{Enc}(k_y^d, K_z^{c\cdot d}))\\
&\quad r_{01} \leftarrow \texttt{Enc}(k_x^c, \texttt{Enc}(k_y^{\neg d}, K_z^{c\cdot \neg d}))\\
&\quad r_{10} \leftarrow \texttt{Enc}(k_x^{\neg c}, \texttt{Enc}(k_y^d, K_z^{\neg c\cdot d}))\\
&\quad r_{11} \leftarrow \texttt{Enc}(k_x^{\neg c}, \texttt{Enc}(k_y^{\neg d}, K_z^{\neg c\cdot \neg d}))\\
&\quad \texttt{return } (r_{00}, r_{01}, r_{10}, r_{11})
\end{aligned}
$$

Above r_{ij} corresponds to the four "garbled rows". Recall that values c, d are "colors", corresponding to the point and permute trick. The above indicates how to garble an AND gate, but any four-input gate can be garbled in an analogous manner.

Notice that the *size* of the garbled truth table is the main cost consideration in GC. These truth tables must be sent from the garbler to the evaluator, so communication cost scales with GC table size, and it turns out that in GC-based protocol communication is typically the bottleneck in overall performance.

Examining the above approach, we can see that the size of each garbled gate is 4λ bits, since there are four encryptions, each of size λ. Today, we will see how to obtain a smaller garbled circuit. In particular, we will ultimately construct a garbling scheme where (1) each garbled AND gate has size 2λ bits, and each garbled XOR gate is "free".

Garbled Row Reduction

To start, let's see how to reduce the size of any garbled gate from 4λ bits to 3λ bits.

For the time being, we will ignore the point and permute trick, and assume that all color bits are zero; we can reintegrate colors later.

Let's start by making explicit our notion of encryption. Let's assume each gate has a distinct id `gid`. We can reformulate a garble AND gate as follows:

$$
\begin{aligned}
&k_z^0, k_z^1 \in_\$ \{0,1\}^\lambda \\
&r_{00} \leftarrow F(k_x^0, (0,0,\texttt{gid})) \oplus F(k_y^0, (0,0,\texttt{gid})) \oplus k_z^0 \\
&r_{01} \leftarrow F(k_x^0, (0,1,\texttt{gid})) \oplus F(k_y^1, (0,1,\texttt{gid})) \oplus k_z^0 \\
&r_{10} \leftarrow F(k_x^1, (1,0,\texttt{gid})) \oplus F(k_y^0, (1,0,\texttt{gid})) \oplus k_z^0 \\
&r_{11} \leftarrow F(k_x^1, (1,1,\texttt{gid})) \oplus F(k_y^1, (1,1,\texttt{gid})) \oplus k_z^1
\end{aligned}
$$

Of course, more formally these rows should be randomly permuted according to point and permute. Assuming the evaluator obtains some appropriate combination of input keys, she can decrypt the corresponding output key.

The *garbled row reduction* technique observes that there is a degree of freedom in the above construction: the gate output keys are simply drawn uniformly at random. Namely, suppose that rather than setting k_z^0 uniformly, the garbler instead chooses k_z^0 according to the input keys. Namely, let's define k_z^0 as follows:

$$k_z^0 \leftarrow F(k_x^0, (0,0,\texttt{gid})) \oplus F(k_y^0, (0,0,\texttt{gid}))$$

(More formally, we define whichever output key is encrypted in the first row this way. The identity of that key will depend on the color bits of the input keys.) Intuitively, the key k_z^0 is still uniformly random to an evaluator who does not hold either k_x^0 or k_y^0 (because F is a PRF), but now it can be deterministically computed by an evaluator who holds both of those input keys. In other words, we can now observe that the above row r_{00} is now the all-zeros string, so the garbler need not send it!

By choosing one of the output keys this way, the garbler has reduced the size of the garbled truth table from 4λ bits to 3λ bits. Again, the reason this is possible is because we leveraged a degree of freedom in the specification of gate output keys. Unfortunately, we cannot simply define two rows of the garbled truth table to be all zeros this way. The reason for this is that, again, the rows of the truth table should be randomly permuted. Thus, the evaluator should not learn which row holds k_z^1. In other words, we can only set *statically-chosen* rows to all zeros. In general, this currently restricts us to saving at most one row.

This trick is called the GRR3 (three garbled row-reduction) optimization.

Free XOR

In our next optimization, we will consider specifically optimizing *XOR* gates, rather than general gates. One way to understand our goal is to reflect on the GMW protocol. Recall that in that protocol, XOR gates are

somehow much simpler than AND gates. Namely, AND gates required OT, but XOR gates were computed locally. Can we achieve a similar effect for GC?

Indeed we can. The main trick is to take an alternate perspective on wire keys. Recall, each wire x has two keys k_x^0, k_x^1. But put another way, we could say that the wire x has a zero key k_x^0 and a *wire-specific correlation* Δ_x. The wire key k_x^1 is $k_x^0 \oplus \Delta_x$.

Now, the Free XOR trick is to garble circuits such that *all wires use the same global correlation* Δ. Namely, the garbler will assign each wire a uniform zero key k_x^0, and it will *define* the one key as $k_x^1 = k_x^0 \oplus \Delta$. If we consider the evaluator's key at runtime, we can see that she will hold $k_x^0 \oplus x\Delta$, where we overload the name of a wire with its value.

Now, if the garbler wishes to garble an XOR gate $z \leftarrow x \oplus y$, he will simply *define* the output key as follows:

$$k_z^0 \leftarrow k_x^0 \oplus k_y^0$$

At runtime, the evaluator locally XORs her keys and obtains the appropriate output key:

$$k_z^0 \oplus (x \oplus y)\Delta$$

Thus, no communication is required to achieve garbled XOR gates.

Remark 1. *Observe that a zero key k_x^0 is known to the garbler, and that the evaluator obtains $k_x^0 \oplus x\Delta$. Notice that these two values together constitute an XOR secret share $[x\Delta]$. In other words, the parties hold $\{x\}$, where x is authenticated (only) to the garbler. This observation will be incredibly important when we consider the maliciously-secure authenticated garbling technique next time.*

Note that while XOR gates are free, AND gates still require communication. Unfortunately, the Free XOR introduces correlations in wire keys, and this means that we can no longer prove security from the basic assumption that F is a PRF. Instead, we will for now grant ourselves a random oracle H. Using GRR3 (and ignoring point and permute), the garbler now handles AND gates as follows:

$$\begin{aligned}
k_z^0 &\leftarrow H(k_x^0, 0, 0, \texttt{gid}) \oplus H(k_y^0, 0, 0, \texttt{gid}) \\
k_z^1 &\leftarrow k_z^0 \oplus \Delta \\
r_{01} &\leftarrow H(k_x^0, 0, 1, \texttt{gid}) \oplus H(k_y^1, 0, 1, \texttt{gid}) \oplus k_z^0 \\
r_{10} &\leftarrow H(k_x^1, 1, 0, \texttt{gid}) \oplus H(k_y^0, 1, 0, \texttt{gid}) \oplus k_z^0 \\
r_{11} &\leftarrow H(k_x^1, 1, 1, \texttt{gid}) \oplus H(k_y^1, 1, 1, \texttt{gid}) \oplus k_z^1
\end{aligned}$$

Half Gates

For our last trick, we will show how to leverage Free-XOR-style encodings to further reduce the size of AND gates from 3λ bits to 2λ bits.

For this last trick, let us consider a special kind of AND gate. Namely, let us suppose that we have a gate $z \leftarrow x \cdot y$, and suppose that the GC evaluator *knows what x is in cleartext*. We will get rid of this assumption later. It turns out that by using garbled row reduction, it is easy to encode such a gate with *one* garbled row. (Indeed, this is the same trick we saw last time for constructing leaky AND gates.)

The garbler handles this AND gate as follows:

$$\begin{aligned}
k_z^0 &\leftarrow H(k_x^0, \texttt{gid}) \\
k_z^1 &\leftarrow k_z^0 \oplus \Delta \\
r &\leftarrow H(k_x^1, \texttt{gid}) \oplus k_y^0 \oplus k_z^0
\end{aligned}$$

At runtime, the evaluator will hold $k_x^0 \oplus x\Delta$ and $k_y^0 \oplus y\Delta$. Recall that we assumed the evaluator knows x. The evaluator can act conditionally, depending on x. If x is zero, then notice that $z = x \cdot y$ is also zero.

The evaluator simply computes $H(k_x^0, \texttt{gid})$, which is equal to k_z^0 by definition. If x is one, then the evaluator will use the single garbled row r and add in her label for wire y. In particular, she computes:

$$\begin{aligned}
& H(k_x^1, \texttt{gid}) \oplus (k_y^0 \oplus y\Delta) \oplus r \\
&= H(k_x^1, \texttt{gid}) \oplus (k_y^0 \oplus y\Delta) \oplus (H(k_x^1, \texttt{gid}) \oplus k_y^0 \oplus k_z^0) && \text{Defn. } r \\
&= k_z^0 \oplus y\Delta \\
&= k_z^0 \oplus xy\Delta && x = 1 \\
&= k_z^0 \oplus z\Delta && z = xy
\end{aligned}$$

Thus, if the evaluator knows x, then the parties can garble that gate for only a single ciphertext.

The above trick is called a garbled *half AND gate*. It turns out that we can use *two* of these half AND gates to garble a "full" AND gate, where the evaluator knows neither gate input. In particular, let us recall Beaver's trick:

$$xy = (x \oplus \alpha)y \oplus (y \oplus \beta)\alpha \oplus \alpha\beta$$

Notice that this computation involves two AND gates where the left input is masked, and all other computations are XORs (and hence free). Thus, one way to garble an AND gate is as follows:

- The garbler samples a random triple $\alpha, \beta, \alpha\beta$ and the parties construct $\{\alpha, \beta, \alpha\beta\}$. Constructing these shares are free, since the garbler knows the cleartext values.
- The parties use Free XOR to compute from $\{x\}$ and $\{y\}$ masked values $\{x \oplus \alpha\}$ and $\{y \oplus \beta\}$.
- The garbler includes two extra bits to reveal these masked values to the evaluator. (Alternatively, we can choose α and β to be the color bits on wires. This saves two bits per AND gate.)
- The parties use two half AND gates to compute $\{(x \oplus \alpha)y\}$ and $\{(y \oplus \beta)\alpha\}$. This costs two garbled rows.
- The parties use Free XOR to complete the computation:

$$\{xy\} = \{(x \oplus \alpha)y \oplus (y \oplus \beta)\alpha \oplus \alpha\beta\}$$

Thus, we can indeed garble an AND gate for two ciphertexts.

Exercise 1. *Reflect on how a cheating garbler can corrupt this AND gate. What avenues does he have to attack the protocol?*

Exercise 2. ***(Hard.)*** *It turns out that an AND gate can also be garbled for $\approx 3/2\lambda$ bits. Investigate the "three halves" technique [RR21]. There is mounting evidence that this surprising technique is essentially optimal.*

Next Time

Today, we saw ways to optimize the cost of the garbled circuit technique. These techniques help to make garbling a more practical mechanism.

Moreover, the optimizations today establish a connection between garbled wire values and authenticated shares $\{x\}$. Next time, we will leverage this new connection to achieve maliciously secure garbling that prevents the garbler from cheating. Roughly speaking, this will involve using our preprocessing functionality to prevent the garbler from corrupting the garbling of AND gates.

References

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Berlin, Heidelberg, July 2008.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Cham.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Berlin, Heidelberg, April 2015.