Authenticated Garbling

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2025

Last time, we introduced Garbled Circuit optimizations, including the Free XOR technique and the half-gates technique. Today, we will combine those ideas with our preprocessing functionality for authenticated multiplication triples to obtain a maliciously-secure, constant-round 2PC protocol.

Namely, recall that we constructed a preprocessing functionality that performs the following:

- The parties can ask the functionality to deal a random bit, and it will deal $\{\alpha\}$, where α is uniform.
- The parties can ask the functionality to deal a triple, and it will deal $\{\alpha, \beta, \alpha \cdot \beta\}$, where α, β are uniform.

Recall that by $\{x\}$, we mean an XOR secret share $[x \cdot (\Delta_A, \Delta_B, 1)]$ where $\Delta_{\{A,B\}} \in \{0,1\}^\lambda$ are party-specific keys.

Today, we will plug this functionality together with GC to obtain a technique called *authenticated garbling*. Our discussion will follow [WRK17a, KRRW18].

Why are Garbled Circuits Insecure in the Malicious Setting?

Recall that GC works between a garbler G and an evaluator E. G "garbles" the circuit and sends it to E, who evaluates by decrypting gates one by one. Informally speaking, GC naturally provide protection against malicious E. The reason for this is that the encryption mechanisms that G uses ensure that E can decrypt only those specific rows that correspond to the input.

Indeed, this restriction on E is essential even in the semi-honest setting. Even in the semi-honest setting, E might try to brute force decrypt all parts of the garbled circuit; if she could, she would learn too much information. Thus, at least informally speaking, GC must be—and is—strong enough to work against a malicious evaluator, even if we are only interested in semi-honest security.

A malicious garbler G, however, can easily break the protocols we have seen so far. As one example, G might garble the circuit incorrectly, causing the evaluator to compute the wrong function, breaking security.

As one trivial example, let us recall Yao's basic construction of a garbled AND gate (ignoring point and permute):

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

A malicious garbler can simply change the logic of this gate. For instance, G might provide the following

encoding instead:

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

Here, G has replaced the agreed-upon AND gate by an OR gate, changing the logic of the function the parties agreed to compute.

Note that this kind of attack is particularly insidious, because essentially the entire point of a garbled circuit construction is that the evaluator cannot tell how the truth table was encrypted. In other words, the very mechanisms that protect an honest G's privacy are preventing E from detecting that the function was incorrectly garbled.

Selective Aborts. One high level insight we can deploy is the following: while E cannot efficiently check that an entire garbled gate is correctly constructed, perhaps it is possible for her to tell that *those parts of the gate she decrypts* were correctly constructed. If those parts are incorrectly constructed, E can abort the protocol. Indeed, this will be one of our main ideas today.

However, this leaves open the possibility of a *selective abort* attack. Indeed, a malicious G can corrupt some parts of a particular garbled gate, then wait to see if E aborts the protocol or not. Depending on whether or not E aborts, G learns something about the input to the corrupted gate.

Sketch of Authenticated Garbling

Based on the above discussion, there are two informal problems that we must solve to achieve a malicious form of garbling:

- **Detecting incorrectly garbled gates.** Somehow, we need to arrange that E can tell when those parts of the GC she evaluates are incorrectly constructed.
- **Preventing selective aborts.** Somehow, we need to prevent G from launching a selective abort attack.

Our approach to solving both of these problems will be by applying authenticated secret shares and our preprocessing functionality. Namely, our main idea will be to propagate the circuit invariant that each wire x holds a doubly-authenticated sharing $\{x\}$. This sharing is authenticated to both G and to E.

This will help E to detect when the circuit is incorrectly garbled, because if G tries to corrupt part of the garbled circuit, and then E evaluates that part, the share out of that part will not be correctly authenticated to E. Thus, E will notice later on that the wire is corrupted, and she will abort.

This still leaves the problem of selective abort. To solve this, we will leverage our ability to sample multiplication triples. The main idea is that we will *not* attempt to prevent the garbler from corrupting specific portions of the garbled circuit. Instead, we will make it such that the event that the evaluator actually *uses* a specific portion of the GC is a *random and input-independent*, from the perspective of G. Formally, since such an event is random from the perspective of malicious G, we can simulate whether or not E aborts by flipping a coin.

Authenticated Half Gates

Recall again the notion of an authenticated secret share $\{x\} = [x \cdot (\Delta_G, \Delta_E, 1)]$. To garble a circuit, we will ensure that each wire x holds $\{x\}$. Matching the Free XOR optimization we saw last time, we will use

$(\Delta_G, \Delta_E, 1)$ as the garbled circuit's global correlation. Thus, XOR gates are easy: G and E simply locally XOR their shares $\{x\}$ and $\{y\}$ to obtain $\{x \oplus y\}$.

Now, let us consider AND gates. In more detail, let us first consider a *half* AND gate $\{x\}, \{y\} \mapsto \{xy\}$ where the evaluator knows (and is convinced of the authenticity of) x. Recall that $\{x\}$ implicity includes a sharing $[x\Delta_G]$. Let X be the garbler's share and let $X \oplus x\Delta_G$ be the evaluator's share.

Let $H : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+1}$ be a random oracle. As we saw with half gates last time, there are two possibilites to consider: Either $x = 0$, or $x = 1$. If $x = 0$, then G's and E's shares match, and both parties can compute $H(X)$. To handle this case, G can simply set his output share as $Z = H(X)$, and E can compute a matching share Z.

If instead $x = 1$, then the output of the gate should instead be $\{xy\} = \{y\}$. Thus, G will provide a single ciphertext that allows E to convert her share on the y wire into a share on the z wire. In particular, G sends:

$$r = H(X \oplus \Delta_G) \oplus Z \oplus Y$$

If $x = 1$, E, who holds $X \oplus \Delta_G$, computes:

$$H(X \oplus \Delta_G) \oplus (Y \oplus y \cdot (\Delta_G, \Delta_E, 1)) \oplus r$$

It is not hard to see that terms in r and E's calculation cancel, yielding the following:

$$Z \oplus Y \oplus y \cdot (\Delta_G, \Delta_E, 1)$$

This share correctly matches G's, and the parties indeed hold $\{xy\}$.

Thus, just like in the semi-honest half-gates construction the garbler can corrupt the garbled truth table row r. But if G does this and E uses it, E's output share will not be authenticated. Thus, if G later tries to reveal that output to E, then she will abort.

Still, notice that G can cause E to *selectively* abort the protocol, depending on the value of x. We address this next.

Combining half gates. The above half gate requires one input be known to E. We now combine two such gates to compute $\{x\}, \{y\} \mapsto \{xy\}$ where both x and y are secret. As with last time, we will leverage the following equation to combine two half gates:

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

In particular, the parties will use the preprocessing functionality to sample a triple $\{\alpha, \beta, \alpha\beta\}$. Note crucially that *neither party* knows α nor β.

So, the first step will be to first compute $\{x \oplus \alpha\}$ and $\{y \oplus \beta\}$. As always, this is free and local. From here, G will reveal both $x \oplus \alpha$ and $y \oplus \beta$ to E. Revealing a value requires sending $\lambda + 1$ bits.

Exercise 1. *Assuming we have a random oracle, There exists a simple optimization that allows each revealed value to incur only (amortized) one bit of communication. Try to see why.*

Note that if E's share was corrupted by some earlier corrupted garbled gate, she will now abort, since her MAC key verification will not pass.

From here, the parties will use two half gates to separately compute the following:

$$\{x \oplus \alpha\}, \{y\} \mapsto \{(x \oplus \alpha)y\}$$
$$\{y \oplus \beta\}, \{\alpha\} \mapsto \{(y \oplus \beta)\alpha\}$$

Again, in each of these half AND gates, G can selectively abort the protocol, depending on $x \oplus \alpha$ (resp. $y \oplus \beta$). But, here is the key point: if G attempts a selective abort here, he is now "selectively" aborting

depending on a *random variable*. In other words, this is not an attack at all, since we can simulate what G sees!

From here, completing the AND gate is a simple matter of locally XORing terms, and we are done.

One way to understand what is happening here is to compare semi-honest half gates with the above authenticated version. In the semi-honest version, G introduces multiplication triples such that E learns nothing from the revealed input to each half gate; in the authenticated version, G and E *jointly* introduce multiplication triples such that *neither* party learns anything from the revealed input to each half gate. Very roughly, this is like using the preprocessing functionality to allow both G and E to *jointly* garble the circuit.

Since parties can handle both XOR and AND gates, we are essentially done. There is some extra detail needed for handling circuit inputs, but this is the same as in the BDOZ protocol: For each input wire, the parties sample a random bit $\{\alpha\}$, open α to the input party, and the input party derandomizes its value. Outputting values is similarly handled by opening the authenticated shares.

Next Time

We have now seen a constant-round maliciously-secure 2PC protocol for arbitrary circuits. This protocol involved only black-box cryptography, meaning that it is reasonably efficient, and practically speaking (minor optimizations to) it is one of the best protocols we know today. The protocol can also be generalized to any number of parties [WRK17b].

Thus, we have now seen quite an interesting variety of protocols for handling arbitrary computations. However, all protocols we have so far seen have had a distinct weakness: they have all required that we express the desired computation as a circuit.

Next time, we will start exploring ways to go beyond basic circuit models of computation. In particular, we will begin study on a technique called *Oblivious RAM*, which is a fundamental cryptographic technique for obfuscating accesses to a random access memory. This will ultimately enable us to consider richer computational problems.

References

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 365–391. Springer, Cham, August 2018.

[WRK17a] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37. ACM Press, October / November 2017.

[WRK17b] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 39–56. ACM Press, October / November 2017.