IT MACs and Authenticated Secret Shares

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2026

So far in this course, we have used feasibility results to achieve maliciously secure protocols: construct a semi-honest MPC protocol, then utilize commitments and zero knowledge proofs to upgrade to malicious security via the GMW compiler.

While this works, the techniques involved are inefficient, due to the compiler's extensive non-black-box use of cryptographic primitives. If the semi-honest protocol Π involves a call to a cryptographic primitive (e.g., encrypting a message, perhaps as part of an OT), then the code of that cryptographic primitive must be executed in the context of a ZK proof, which is expensive. In addition, the ZK proof system must be invoked to prove validity of *each protocol message*, which is clearly expensive, especially once one considers that validity of a message is defined with respect to the entire protocol transcript up to that point.

Today, we will investigate ways to achieve malicious security while using only black box techniques. The main idea we will consider today is called authenticated secret sharing. At the very highest level, we will today adjust the GMW protocol such that all values are *authenticated* to the parties. We will focus on the two-party setting, or 2PC with a dealer.

Review of the GMW Protocol and its Vulnerability to Malicious Adversaries

Recall that the GMW protocol worked by assigning to each wire in a Boolean circuit an XOR secret share of values. Namely, for each wire x, the parties hold XOR secret shares $[x]$. Shares are propagated through gates as follows:

- **Inputs.** The party whose input corresponds to a particular input wire secret shares their input bit.
- **XOR gates.** Parties simply locally XOR their shares.
- **AND gates.** Parties multiply their secret-shared gate inputs by consuming a secret-shared multiplication triple of form $[\alpha, \beta, \alpha \cdot \beta]$ where $\alpha, \beta \leftarrow \{0,1\}$ are uniform. In particular, parties reveal to themselves $x \oplus \alpha$ and $y \oplus \beta$, and then they can compute $[x \cdot y]$ locally.
- **Output gates.** Parties exchange secret shares and reconstruct the output bit.

The GMW protocol is not robust in the presence of a malicious adversary. It is easy to see this in the two-party case where GMW runs between A and B. Suppose A is corrupted, and consider some sharing $[x] = (x_A, x_B)$. To attack the protocol, A can simply flip her share x_A. Namely, she sets $x_A \leftarrow x_A \oplus 1$. Due to the additively homomorphic property of XOR shares, this has the effect of changing $[x]$ to $[x \oplus 1]$. In other words, an adversarial A can flip bits in the circuit as she pleases. In particular, malicious A could flip the value of an output of the circuit just before it is reconstructed. This is, of course, not allowed by our ideal functionality.

It's worth pointing out that the GMW compiler would indeed rule out such attacks, because "flipping bits" is not prescribed by the protocol, so B will notice that A's output share is not consistent with the protocol. Again, while this works, it is expensive.

Intuitively, the problem with GMW is that there is no mechanism by which B can easily determine whether or not A has flipped a particular bit. Today we will understand the notion of an *authenticated secret share*, which gives a simple mechanism by which to prevent such attacks.

Authenticated Secret Sharing

The main idea of an authenticated secret sharing of a bit x is to associate with the shares a global key $\Delta \leftarrow \{0,1\}^\lambda$. In particular, let us suppose that B holds such a global key $\Delta_B \leftarrow \{0,1\}^\lambda$, and A does not know this key. Now, let us revisit a simple XOR share $[x]$. Again, the problem with such a share is that A can simply flip her share, resulting in a valid XOR sharing $[x \oplus 1]$.

Instead, let us consider secret sharing the vector $[x \cdot (\Delta_B, 1)]$. That is, A and B respectively hold strings $X_A, X_B \in \{0,1\}^{\lambda+1}$ s.t. $X_A \oplus X_B = x \cdot (\Delta_B, 1)$. Before considering dishonest A, let's start by an honest interaction, where A wishes to open her share to B. The procedure is as follows:

- A sends her share X_A to B.
- B XORs the least significant bits $\texttt{lsb}(X_A) \oplus \texttt{lsb}(X_B)$ and calls the result x. Note that if A is indeed honest, this quantity is x by construction.
- Next, B XORs the leading λ bits $\texttt{msbs}(X_A) \oplus \texttt{msbs}(X_B)$. Note that if A is honest, this quantity should be $x \cdot \Delta_B$. That is, this quantity is either 0^λ, if $x = 0$, or it is Δ_B. B checks that this is true: namely, he performs the following equality check:

  $$\texttt{msbs}(X_A) \oplus \texttt{msbs}(X_B) \stackrel{?}{=} x \cdot \Delta_B$$

  (Recall, Δ_B is B's key, so he knows it.) If the above equality holds, B is convinced that A's share is correctly constructed; if it does not hold, B can abort.

Now, suppose parties hold $[x \cdot (\Delta_B, 1)]$ and a malicious A would like to reveal to B the flipped bit $x \oplus 1$. Clearly A can easily flip the least significant bit of her share, which flips x. However, if A does *only* this, B will abort. Indeed, in the above protocol, the equality check will not pass.

So how can cheating A win this game? A winning A must send some X'_A whose MAC part is consistent with $x \oplus 1$, i.e.

$$\texttt{msbs}(X'_A) \oplus \texttt{msbs}(X_B) = (x \oplus 1) \cdot \Delta_B.$$

But all A holds is her honestly generated X_A, for which

$$\texttt{msbs}(X_A) \oplus \texttt{msbs}(X_B) = x \cdot \Delta_B.$$

XORing the two conditions gives $\texttt{msbs}(X_A) \oplus \texttt{msbs}(X'_A) = \Delta_B$. So producing X'_A is equivalent to guessing Δ_B, which succeeds with probability $2^{-\lambda}$.

In other words, A's share of $[x \cdot (\Delta_B, 1)]$ is *authenticated* to B. In particular, this share is authenticated via a so-called IT-MAC (information-theoretic message authentication code); A cannot change her share without winning some information-theoretic game, by simply guessing Δ_B.

BDOZ-Style Sharings. So far, the sharing of x is authenticated only to B. I.e., B has security against a cheating A, but not vice versa. This is easy enough to remedy: We will authenticate every bit with *two* keys. Namely, if parties hold secret shares of x, we will arrange that they hold $[x \cdot (\Delta_A, \Delta_B, 1)]$. Now, the shares are authenticated to *both* parties. For convenience, we will denote such a share by $\{x\}$. We also extend this notation to vectors of bits in the natural manner. Namely $\{x, y\}$ denotes a pair of authenticated bits $\{x\}, \{y\}$.

Remark 1. *It is clear that this notion generalizes from two parties to p parties. Namely:*

$$\{x\} = [x \cdot (\Delta_0, ..., \Delta_{p-1}, 1)]$$

Notice that such shares have size $O(p\lambda)$ bits.

This style of sharing is often called a BDOZ-style share, after [BDOZ11] who showed how to use such sharings to construct maliciously-secure protocols. (Note that these are different in form than so-called SPDZ-style shares [DPSZ12], which we will investigate later.)

Importantly, notice that, similar to standard XOR shares, BDOZ-style sharings are XOR homomorphic: In particular:

$$\begin{aligned}
\{x\} \oplus \{y\} &= [x \cdot (\Delta_A, \Delta_B, 1)] \oplus [y \cdot (\Delta_A, \Delta_B, 1)] \\
&= [x \cdot (\Delta_A, \Delta_B, 1) \oplus y \cdot (\Delta_A, \Delta_B, 1)] && \text{XOR sharing homomorphism} \\
&= [(x \oplus y) \cdot (\Delta_A, \Delta_B, 1)] && \text{AND distributes over XOR} \\
&= \{x \oplus y\}
\end{aligned}$$

To open a sharing $\{x\}$ to a particular party, say A, B sends their shares of $[x]$ and their shares of $[x \cdot \Delta_A]$ (but not their share of $[x \cdot \Delta_B]$!) to that party. A then checks the opened shares are consistent; if not, A can conclude that B must be cheating and abort.

One final important detail is that if all parties agree on some constant c, they can construct a sharing $\{c\}$ for free. Namely, party A takes as its share $[c \cdot \Delta_A, 0^\lambda, c]$ and B as its share $[0^\lambda, c \cdot \Delta_B, 0]$. These shares by construction XOR to $c \cdot (\Delta_A, \Delta_B, 1)$, as required.

MPC from BDOZ-Style Sharings

Now, let's use BDOZ sharings to construct a maliciously-secure protocol. To do this, we will revisit a high level concept from the start of this course: we will assume access to a third party who will compute for us *correlated randomness*. We will show how to securely instantiate such a third party later.

Formally, we will assume a *preprocessing functionality*. This functionality first deals keys Δ_A to A and Δ_B to B. Then, it can be used to generate two forms of correlation:

- **Authenticated bits.** The functionality distributes random, shared, authenticated bits of form $\{\alpha\}$ where $\alpha \leftarrow \{0,1\}$. Notice that no party knows α, and the sharing of it is authenticated to everyone.
- **Authenticated triples.** The functionality distributes random, shared, authenticated triples of form $\{\alpha, \beta, \alpha \cdot \beta\}$ where $\alpha, \beta \leftarrow \{0,1\}$. Notice that no party knows $\alpha, \beta, \alpha \cdot \beta$, and their sharings are authenticated to everyone.

Exercise 1 (A corrupt party's key)**.** *In the preprocessing functionality, a corrupt A may be allowed to choose her own Δ_A, even $\Delta_A = 0^\lambda$, delivering it to the functionality instead of receiving it. Show that this does not help her attack B, and say precisely what she loses by doing so.*

Now, given such a preprocessing functionality, we can construct a maliciously-secure 2PC protocol. The protocol will proceed almost identically to the GMW protocol. In particular, the parties will consider a Boolean circuit C, and the main invariant will be that for each circuit wire x, the parties will hold $\{x\}$.

Formally, the parties proceed as follows:

- **Input wires.** Suppose this input wire belongs to some party, say A, who holds corresponding input bit x. The parties use the preprocessing functionality to construct a random authenticated bit $\{\alpha\}$. They reveal α to (only) A. (If A detects any cheating in this step, she aborts.) A then computes $x \oplus \alpha$ and sends it to B. Parties use BDOZ-style sharing's support for constants to compute $\{x \oplus \alpha\}$, and then they locally XOR:
$$\{\alpha\} \oplus \{x \oplus \alpha\} = \{x\}$$
This establishes our invariant on this input wire.
- **XOR gates.** By induction, suppose parties already hold $\{x\}$ and $\{y\}$. They locally add their shares to compute $\{x \oplus y\}$.
- **AND gates.** By induction, suppose parties already hold $\{x\}$ and $\{y\}$. They use the preprocessing functionality to construct a random authenticated multiplication triple $\{\alpha, \beta, \alpha \cdot \beta\}$. Next, the parties locally compute $\{x \oplus \alpha\}$ and $\{y \oplus \beta\}$, and then they reveal $x \oplus \alpha$ and $y \oplus \beta$ to themselves. If any party detects cheating, they abort. Otherwise, they compute:
$$(x \oplus \alpha) \cdot \{y\} \oplus (y \oplus \beta) \cdot \{\alpha\} \oplus \{\alpha \cdot \beta\} = \{x \cdot y\}$$

- **Output wires.** By induction the parties hold $\{x\}$. They exchange shares to reconstruct x. If any party detects cheating, they abort.

Exercise 2 (Arithmetic circuits)**.** *The above protocol can be naturally generalized from a circuit whose wires hold bits to one whose wires hold arithmetic values, where gates compute addition and multiplication rather than XOR and AND. Think through what changes need to be made to the representation of shares, the preprocessing functionality, and the protocol itself to support this change. For which sets of arithmetic values does the change work? For which does it not, and why?*

In short, the malicious security of this protocol is immediate from the security of the authenticated secret sharing and the assumption that the preprocessing functionality is secure. A bit more formally, simulation security against any corrupted party, say A, could be shown as follows:

- The simulator interacts with the adversary and the MPC functionality. It tries to "trick" the adversary into thinking it is in the real-world protocol. To do this, the simulator plays the role of (1) the preprocessing functionality and (2) honest B.
- The simulator can extract the adversary's input: it plays the preprocessing functionality, so it knows the mask α on each of A's input wires, and A must send $x \oplus \alpha$ (see the input-wire step). Thus by the time all input wires are handled (if no abort occurred), the simulator knows A's input, which it hands to the ideal functionality.
- From here, the simulator just runs the protocol as if it is the honest parties, albeit with arbitrarily-chosen input (say, the all-zeros input).
- Finally, at output wires, the simulator uses its knowledge of the functionality output to forge B's opened share. It can do so because it knows Δ_A (it dealt the keys), so it can produce a share that passes A's check for any desired output bit.

Of course, the simulator must also account for aborts, but this is easily handled by checking whether the adversary sends ill-formed shares, and appropriately sending either `abort` or `continue` to the ideal MPC functionality.

Exercise 3 (The full simulator)**.** *If you are still having trouble with simulation, it is a good exercise to formally write out a simulator for a corrupt A in full, including abort handling. In particular: at an output wire, A may receive B's opened share and then abort before sending her own. Show that the simulator can handle this using the fact that the ideal functionality delivers output to the adversary before honest parties. Does it matter whether A or B opens the output first? Justify your answer.*

Next Time

We have now described a maliciously-secure 2PC protocol that uses only black box cryptography. However, this protocol assumed access to an ideal preprocessing functionality for generating correlated randomness. Next time, we will see how to instantiate this preprocessing functionality efficiently [NNOB12].

References

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Berlin, Heidelberg, May 2011.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Berlin, Heidelberg, August 2012.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Berlin, Heidelberg, August 2012.