IT MACs and Authenticated Secret Shares

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2025

Last time, we discussed how to construct a Zero Knowledge (ZK) proof system from a semi-honest-secure MPC protocol. In particular, we examined the MPC-in-the-head paradigm [IKOS07] through the lens of the ZKBoo construction [GMO16]. Recall that ZK proof system can be used to transform a semi-honest secure protocol into a maliciously-secure one, via the GMW compiler [GMW87]. In short, the parties commit to their inputs and randomness, then at every protocol step they prove in ZK that their sent message is constructed according to the protocol specification.

Thus, we now have feasibility results for generic p-party MPC, even when $p-1$ parties collude and try to actively break the protocol. Even in this case, the extent of the adversary's capability is formalized by a simple ideal functionality: the adversary can (1) abort the protocol, or (2) deliver output as intended.

However, our techniques are currently quite inefficient, due to the use of the GMW compiler. In particular, the GMW compiler involves 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 extremely expensive. In addition, the ZK proof system must be invoked to prove validity of *every single message*, which is clearly expensive.

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 each of the parties.

Review of the GMW Protocol and its Vulnerability to Malicious Adversaries

Recall that the GMW protcol worked by assigning to each wire in a Boolean circuit an XOR secret share of values. Namely, for each wire x, the p 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 $\alpha, \beta \in_{\$} \{0,1\}$ are uniform. In particular, parties reveal to themselves $x \oplus \alpha$ and $y \oplus \beta$, and then the can compute $[x \cdot y]$ locally.
- **Output gates.** Parties exchange secret shares and reconstruct the output bit.

The GMW protocol is not robust 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. Authenticated secret sharing is one mechanism by which we can prevent A from performing such an attack.

Authenticated Secret Sharing

The main idea of an authenticated secret sharing of a bit x is to associate with the shares a global key $\Delta \in_\$ \{0,1\}^\lambda$. In particular, let us suppose that B holds such a global key $\Delta_B \in_\$ \{0,1\}^\lambda$, 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 $\mathtt{lsb}(X_A) \oplus \mathtt{lsb}(X_B)$ 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 $\mathtt{msbs}(X_A) \oplus \mathtt{lsb}(X_B)$. 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:

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

  (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 holds $[x \cdot (\Delta_B, 1)]$ and a malicious A would like to reveal to B the flipped bit $x \oplus 1$. Now, 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? In short, a winning A must somehow send shares consistent with $(x \oplus 1) \cdot \Delta_B$, which means that A must somehow guess Δ_B. But $\Delta_B \in \{0,1\}^\lambda$ is a long string, so guessing will only succeed with negligible probability.

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; 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.

In general, we can set up a p-party generalization of such sharings, where a particular bit share is authenticated to *each* of the parties. For convenience, we will denote such a share by $\{x\}$. Namely:

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

We also extend this notation to vectors of bits in the natural manner. Namely $\{x, y\}$ denotes a pair of authenticated bits $\{x\}, \{y\}$.

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:

$$\begin{aligned}
\{x\} \oplus \{y\} &= [x \cdot (\Delta_0, ..., \Delta_{p-1}, 1)] \oplus [y \cdot (\Delta_0, ..., \Delta_{p-1}, 1)] \\
&= [x \cdot (\Delta_0, ..., \Delta_{p-1}, 1) \oplus y \cdot (\Delta_0, ..., \Delta_{p-1}, 1)] && \text{XOR sharing homomorphism} \\
&= [(x \oplus y) \cdot (\Delta_0, ..., \Delta_{p-1}, 1)] && \text{AND distributes over XOR} \\
&= \{x \oplus y\}
\end{aligned}$$

To open a sharing $\{x\}$ to a particular party P_i, the parties will send their shares of $[x]$ and their shares of $[x \cdot \Delta_i]$ to that party. Party P_i then checks the opened shares are consistent, and if not they can abort. Notice that we *do not* send to party P_i the shares of other party MACs $[x \cdot \Delta_{j \neq i}]$, as this would reveal Δ_j to P_j, which is not secure.

One final important detail is that if all parties agree on some constant c, they can construct a sharing $\{c\}$ for free. Namely, party P_0 takes as its share $[c \cdot \Delta_0, ..., 0^\lambda, 1]$ and each party P_i takes as its share $[0, ..., c \cdot \Delta_i, ..., 0]$. These shares by construction XOR to $c \cdot (\Delta_0, ..., \Delta_{p-1}, 1)$, 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 deals keys Δ_i to each party P_i, and then it can be used to generate two forms of correlation:

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

Now, given such a preprocessing functionality, we can construct a maliciously-secure p-party MPC 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 P_i who holds corresponding input bit x. The parties use the preprocessing functionality to construct a random authenticated bit $\{\alpha\}$. They reveal α to (only) P_i. (If P_i detects any cheating in this step, she aborts.) P_i then computes $x \oplus \alpha$ and broadcasts this bit to all other parties. 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.

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 subset of parties 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) all honest parties.
- The simulator can extract the input from the adversary since (1) the simulator knows the adversary's keys Δ_i (indeed, the simulator plays the preprocessing functionality, so it gives keys to the adversary) and (2) the adversary must send bits $x \oplus \alpha$ corresponding to its input (see **Input wires** description). Thus, by the time all input gates have been handled, (if no abort occurred) the simulator "knows" the adversary's input, which it can hand to the ideal MPC 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 shares that successfully decode to the desired output.

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.

Next Time

We have now described a maliciously-secure MPC protocol that uses only black box cryptography. However, this protocol assumed access to an ideal preprocessing functionality for generating correlated randomness. In addition, the protocol has high round complexity, just like the GMW protocol. In upcoming lectures, we will address both of these problems. First, how can we instantiate the preprocessing functionality and second, how can we adapt the constant-round garbled circuit technique to work in a malicious manner.

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.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.