Towards Authenticated Triples

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2026

Last time, we discussed notions of BDOZ-style *authenticated shares* and *information-theoretic* MACs. We used these concepts to construct a maliciously-secure MPC scheme. The basic idea was to use the GMW protocol, but to (1) replace all simple XOR shares of values by authenticated shares and (2) replace XOR-shared AND triples by authentically-shared triples.

Recall that in the two party setting, we defined an authenticated share of a bit $x \in \{0,1\}$ as a sharing $[x \cdot (\Delta_A, \Delta_B, 1)]$ where $\Delta_A \in \{0,1\}^\lambda$ (resp. $\Delta_B \in \{0,1\}^\lambda$) is a global key held by A (resp. B). Recall, we wrote such shares as $\{x\}$. The crucial insight was that because A does not know Δ_B (resp. B does not know Δ_A), even a malicious adversary cannot "flip a sharing", because attempting to do so amounts to guessing Δ_B (resp. Δ_A), which is hard because Δ_B is random.

Based on this insight, we upgraded the GMW protocol with malicious security by placing authenticated sharings on every wire of the circuit C. However, our protocol was constructed by assuming access to a particularly powerful *preprocessing functionality*. Indeed, we assumed a preprocessing functionality that performs the following tasks:

- **Initialization.** Upon initialization, the functionality samples random $\Delta_A, \Delta_B \leftarrow \{0,1\}^\lambda$ and then sends these to A (resp. B).
- **Random bits.** The parties may query the functionality for a random bit. In this case, the functionality flips a coin $\alpha \leftarrow \{0,1\}$, samples an authenticated sharing $\{\alpha\}$, and deals shares to the parties.
- **Random triples.** The parties may query the functionality for a multiplication triple. In this case, the functionality samples two bits $\alpha, \beta \leftarrow \{0,1\}$, samples authenticated sharings $\{\alpha, \beta, \alpha \cdot \beta\}$, and deals shares to the parties.

This preprocessing functionality was sufficient to achieve MPC, but we have not yet shown how to construct it.

Remark 1. *The above functionality is, in fact, a bit stronger than what researchers often consider. More typically, a preprocessing functionality will include some combination of the following clarifications:*

- *If a party is corrupted, then the adversary may choose that party's randomness on behalf of the functionality. In particular, the above functionality generates random sharings, e.g.* $\{r\} = [r \cdot (\Delta_A, \Delta_B, 1)] = (R_A, R_B)$ *where* $R_A \oplus R_B = r \cdot (\Delta_A, \Delta_B, 1)$. *This clarification on the functionality states that if the adversary corrupts A (resp. B), then the adversary is allowed to choose R_A (resp. R_B), and then the functionality will sample R_B (resp. R_A) such that the above equality holds. Similarly, the adversary who corrupts A (resp. B) is allowed to choose Δ_A (resp. Δ_B). In other words, the functionality is* endemic.
- *The functionality provides an additional "global-key query" interface to the adversary, where a corrupted A may guess a bit of key Δ_B (and vice versa). In particular, A submits a guess that the i-th bit of Δ_B is b. If this guess is correct, the functionality informs the adversary. Otherwise, the functionality sends ABORT to all parties and halts.*

Both of the above clarifications weaken the preprocessing functionality. The reasons for weakening the functionality in this way are twofold:

- *First, this weaker functionality is* still sufficient *to achieve MPC. The ability to choose randomness does not help the adversary learn the honest party's randomness, and guessing Δ_B is still hard.*
- *Second, this weaker functionality is* easier to achieve*. Indeed, the techniques we will study today achieve this kind of weaker functionality.*

Exercise 1. *The functionality allows a corrupt receiver to guess Δ bit by bit, aborting on a wrong guess. Argue that she learns k bits with probability at most 2^{-k}, and that this does not help her forge a MAC.*

Today, our goal will be to start demystifying the above preprocessing functionality in the two-party setting. This preprocessing functionality will be built in much the same way as in the semi-honest setting: we will use OT to construct multiplication triples. We will in particular look at two steps:

- How can we upgrade OT with malicious security?
- How can we use malicious OT to construct multiplication triples.

While we will be investigating *specific techniques*, our emphasis will be on *high level concepts*; namely, what tools do we have, in general, for achieving malicious security.

Today, we will not have time to construct the full preprocessing functionality, but we will have time to achieve its first capability — generating authenticated random bits.

Malicious Oblivious Transfer

Recall that we have studied two OT objects: base OT protocols and *OT extension* [IKNP03]. Today, we will focus on how to augment OT extension with malicious security.

For base OTs, we will simply state that the parties use the GMW compiler to achieve malicious security. Namely, they run a semi-honest OT protocol with interleaved ZK proofs. This is sufficient for us for two reasons:

- First, since we only need a small number of base OTs to bootstrap our way to any secure computation, we care much less about the performance of these base OTs.
- Second, many malicious OT protocols have similar flavor to indeed invoking the GMW compiler, except that they use some custom—and hence more efficient—proof technique, rather than employing generic ZK.

As we will see next, the IKNP OT extension protocol we studied previously can also be made maliciously secure, with low overhead.

Recall that IKNP uses a small number λ of base OTs to generate a large number n of extended OTs. In particular, recall the technique used base OTs to generate the following secret-shared matrix:

$$[r \otimes \Delta]$$

Here, $r \in \{0,1\}^n$ is the receiver's string of (random) choice bits, and Δ is the sender's key. This is a tall, thin matrix which we construct column by column. Recall that each *row* of the matrix can be understood as an extended OT. Indeed, if the sender holds $R_i \in \{0,1\}^\lambda$, then the receiver must hold:

$$R_i \oplus r_i\Delta = \begin{cases} R_i & \text{if } r_i = 0 \\ R_i \oplus \Delta & \text{otherwise} \end{cases}$$

To achieve true OT, we can use a random oracle to break the correlations, but actually this functionality—where parties obtain shares related by some global correlation Δ—is in fact exactly what we want. This form of OT (where all pairs of messages are related by a fixed Δ) is called **correlated OT**.

The above OT protocol is semi-honest secure, but it is not yet malicious secure, even if the base OTs *are* malicious secure. Recall that in each base OT, the parties flip roles. The OT receiver acts as the OT sender, and she is supposed to send two messages m_0, m_1 that differ precisely by her choice bits r, i.e. $m_0 \oplus m_1 = r$. The problem is that there is so far nothing enforcing that the receiver use the *same* difference r in each base

OT. Ultimately, a carefully crafted attack will allow a receiver to learn bits about Δ by simply guessing them, then observing S's behavior in some outer protocol.

What is needed is some kind of *consistency check* by which the OT sender can become convinced that the OT receiver used the same vector $r \in \{0,1\}^n$ in each of the λ base OTs. (Only the receiver needs checking: the sender's only messages in IKNP are its base-OT choice bits, and it may choose Δ freely, so a malicious sender is handled by the malicious base OTs.) We will describe the main ideas from [KOS15].

The main tool we will use to upgrade IKNP with malicious security is the ability to **take random linear combinations of messages**. This technique is a common and powerful tool that we can use to subvert malicious adversaries.

In particular, after the parties (supposedly) compute $[r \otimes \Delta] = (M^S, M^R)$, S samples and sends to R a random challenge string $\chi \leftarrow \{0,1\}^n$.

Again, for each row i, if R acted honestly, then it should be that $M_i^S \oplus M_i^R = r_i \cdot \Delta$. Recall that R knows r and S knows Δ. In short, the parties use the challenge string χ to compute *random linear combinations* of information they hold. Namely, R computes and sends to S two values:

$$\bigoplus_i \chi_i \cdot M_i^R \qquad \bigoplus_i \chi_i \cdot r_i$$

S checks the following:

$$\left(\bigoplus_i \chi_i \cdot M_i^R\right) \oplus \left(\bigoplus_i \chi_i \cdot M_i^S\right) = \left(\bigoplus_i \chi_i \cdot r_i\right) \cdot \Delta$$

If this check fails, then S aborts.

The first insight is that if R is honest, then this check will indeed pass:

$$\begin{aligned}\left(\bigoplus_i \chi_i \cdot M_i^R\right) \oplus \left(\bigoplus_i \chi_i \cdot M_i^S\right) &= \left(\bigoplus_i \chi_i \cdot (M_i^R \oplus M_i^S)\right) \\ &= \left(\bigoplus_i \chi_i \cdot (r_i \cdot \Delta)\right) \\ &= \left(\bigoplus_i \chi_i \cdot r_i\right) \cdot \Delta\end{aligned}$$

The second insight is that if R tries to cheat and incorrectly guesses some bit of Δ, then the above consistency check will cause S to abort with probability $\geq 1/2$. Indeed, suppose that R corrupts some single row i such that $M_i^R \oplus M_i^S \neq r_i \cdot \Delta$. Now, suppose that S asked R to prove that the above equality instead held. This would involve guessing some bit of Δ, similar to our reasoning about authenticated MACs. If R guesses wrong, then the check fails, and S aborts. Note that if R does indeed correctly guess a bit of Δ, then they will not be caught by this consistency check, but also note that they cannot guess *all* bits of Δ—only some small number before being caught with high probability—and hence they cannot guess S's input to the random oracle.

Similarly, if any invalid corrupted row is included in the above sum check, then S will abort with probability 1/2.

To achieve overwhelming security for S, we can simply repeat the above consistency check λ times.

There is one small remaining problem, which is that this consistency check introduces a privacy problem for the *receiver*. Indeed, each random linear combination leaks one bit of information about the string r. The solution is to pad: rather than generating n OTs, the parties generate $n + \lambda$ and include them all in each consistency check, then discard the last λ. For each ℓ-th challenge $\chi^{(\ell)}$, S uniformly samples a length-n challenge, then concatenates a one-hot encoding of ℓ to the end, yielding a length $n + \lambda$ challenge. The point here is that the single extra burned OT acts as a one-time pad, masking any leakage.

Exercise 2. *Suppose the parties skipped the padding and included only the n real OTs in the checks. What exactly does S learn about r?*

To reiterate, the main idea to upgrade OT extension is to incorporate a *consistency check* based on computing *random linear combinations* of elements.

From OT to Authenticated Bits

Now that we have a correlated OT protocol, we can consider trying to instantiate our preprocessing functionality. Our discussion will follow the preprocessing protocol discussed here [NNOB12]. As a first task, let's consider how to sample a random bit $\{\alpha\} = [\alpha \cdot (\Delta_A, \Delta_B, 1)]$.

To do so, notice that parties can run two instances of correlated OTs such that for each α, A and B run OT, ultimately resulting in $[\alpha_A \cdot \Delta_B]$ and $[\alpha_B \cdot \Delta_A]$, where α_A, Δ_A are known to A (resp. for B). Notice that A can locally extend her share to $[\alpha_A \cdot (\Delta_A, \Delta_B, 1)] = \{\alpha_A\}$, and similarly for B and $\{\alpha_B\}$. We can define $\alpha = \alpha_A \oplus \alpha_B$, and the parties can compute $\{\alpha\} = \{\alpha_A\} \oplus \{\alpha_B\}$. Hence, once we have a maliciously-secure correlated OT protocol, it is easy to obtain authenticated random bits.

Of course, parties should ideally generate such random bits in *large batches*, to take advantage of OT extension.

Next Time

We have now seen how to upgrade OT extension with malicious security, and we have seen how to generate from correlated OT authenticated bits. Next time, we will use our tooling to also obtain authenticated multiplication triples. Later, we will also see how to use them to obtain a malicious notion of garbled circuits.

References

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Berlin, Heidelberg, August 2003.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Berlin, Heidelberg, August 2015.

[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.