Correlated Randomness

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2026

Last time, we formalized our first notion of security: semi-honest security (for deterministic functionalities). Recall that this notion captures an adversary who is "honest but curious". Namely, the adversary follows the protocol as we describe it but is eager to learn as much as possible from the protocol transcript. A protocol Π is secure if that curious adversary learns nothing, except what can be learned from the input/output behavior of the securely computed program. Remember that we prove Π satisfies such a notion of security by constructing *simulators*: programs that reorganize information seen in an ideal world such that the reorganized information "looks like" information in the real world. This notion of security is powerful, but so far we have only seen contrived protocols that satisfy it.

Recall that one of our contrived protocols allowed two parties to securely compute XOR:

Protocol 1 (Two-Party XOR)**.** *The protocol runs between two parties A and B.*

- ***Input:*** *Party A inputs a bit $x \in \{0,1\}$; B inputs a bit $y \in \{0,1\}$.*
- ***Output:*** *Both parties output $x \oplus y$.*
- ***Procedure:***
  - *A sends x to B.*
  - *B sends y to A.*
  - *A (resp. B) locally computes $x \oplus y$ and outputs the result.*

Perhaps surprisingly, achieving a seemingly-similar computation—where the two parties AND their input bits—is much harder.

Exercise 1. *Reflect on why Protocol 1 does not work when we replace logical XOR by logical AND.*

Today, we will see our first MPC protocol that is in some sense nontrivial. Namely, we will construct a protocol Π that securely computes AND. Today, we will achieve this functionality by making an additional assumption: Our protocol Π will run between three parties, and it will be semi-honest secure, assuming that no two parties collude. One of these parties will serve as a *dealer*. The dealer will have no input and no output, and will simply aid the other two parties in computing AND. Note that this assumption that no two parties collude is undesirable, but it will allow us to construct a very simple and efficient protocol. Later, we will see ways to remove the third party.

Exercise 2. *What is the difference between (1) assuming a dealer and (2) assuming a trusted third party, as defined last time?*

Along the way to constructing Π, we will introduce two concepts that are central to the study of secure computation:

- Secret sharing.
- Correlated randomness.

Secret Sharing

Let's consider a contrived scenario. A committee consisting of n parties have some important secret $x \in \{0,1\}^\ell$ that they would like to keep; perhaps x is a key used for some important purpose. The secret x is so sensitive that the n parties do not wish to store it in any one place. Somehow, they wish to split the secret x into n pieces, then distribute one piece to each of themselves. Our n parties want the following informal properties of their pieces:

- **Reconstructible:** If the n parties come together, they can use their n pieces to efficiently recover the secret x.
- **Security:** If an adversary manages to steal only a (proper) subset of the pieces—even up to $n-1$ pieces—then they should learn nothing about x.

What our n parties wish for is a simple variant of a *secret-sharing scheme*. In particular, this variant is called an n-out-of-n secret sharing scheme, because all n shares are required to reconstruct the secret. (One can also consider a t-out-of-n secret-sharing scheme where $t < n$ shares allow one to reconstruct the secret.)

Definition 1 (n-out-of-n secret sharing scheme)**.** *An n-out-of-n secret sharing scheme is a pair of efficient algorithms* $\texttt{Share}, \texttt{Reconstruct}$*:*

- $\texttt{Share}$ *is a randomized algorithm taking as input a length-ℓ string $x \in \{0,1\}^\ell$ and outputting n shares $x_0, ..., x_{n-1} \in \{0,1\}^\ell$.*
- $\texttt{Reconstruct}$ *is a deterministic algorithm taking as input n length-ℓ shares $x_0, ..., x_{n-1} \in \{0,1\}^\ell$ and outputting a string $y \in \{0,1\}^\ell$.*

These algorithms should satisfy the following properties:

- ***Correctness:*** *Sharing a secret and then reconstructing that secret should satisfy the obvious correctness property, where the secret x is indeed reconstructed:*

$$\Pr\left[\, y = x \;\middle|\; \begin{array}{l} (x_0, ..., x_{n-1}) \leftarrow \texttt{Share}(x) \\ y := \texttt{Reconstruct}(x_0, ..., x_{n-1}) \end{array} \right] = 1$$

- ***Security:*** *Any proper subset of shares should hide the secret. Formally, let $x, x' \in \{0,1\}^\ell$ be two arbitrary secrets, and let $I \subsetneq [n]$ be a proper subset of the indices $0, ..., n-1$. The following distributions are identical:*

$$\{\ \{x_i\}_{i \in I} \mid (x_0, ..., x_{n-1}) \leftarrow \texttt{Share}(x)\ \} \equiv \{\ \{x_i\}_{i \in I} \mid (x_0, ..., x_{n-1}) \leftarrow \texttt{Share}(x')\ \}$$

While the above definition might appear quite powerful, it is very easy to construct. In particular, the following scheme, which just flips coins and computes XOR, straightforwardly satisfies the definition:

Definition 2 (XOR Secret Sharing Scheme)**.**

- *To* $\texttt{Share}$ *some secret x, we choose $n-1$ uniformly random shares $x_1, ..., x_{n-1} \leftarrow \{0,1\}^\ell$, and we set x_0 such that the shares properly reconstruct to x:*

$$x_0 = x \oplus \left(\bigoplus_{i \in \{1,...,n-1\}} x_i \right).$$

- *To* $\texttt{Reconstruct}$ *shares, we simply XOR them together.*

This scheme is correct by construction: we specifically choose the shares such that they reconstruct to x. Moreover, security is also immediate: even $n-1$ shares leak nothing about x, because the final remaining share can fully "explain" any secret x. Namely, given some subset of shares $\{x_i\}_{i \in I}$, for every secret x there exists some remaining share y such that

$$y \oplus \bigoplus_{i \in I} x_i = x.$$

A bit more formally, security follows from the classic one-time pad property:

Fact 1 (One-Time Pad)**.** *Let $x \in \{0,1\}^\ell$ be an arbitrary string. The following distributions are identical:*

$$\left\{ \; x \oplus \alpha \;\middle|\; \alpha \leftarrow \{0,1\}^\ell \; \right\} \equiv \left\{ \; \alpha \;\middle|\; \alpha \leftarrow \{0,1\}^\ell \; \right\}$$

That is, $x \oplus \alpha$ is identically distributed to a uniform string, even though the sum involves an arbitrarily distributed string x.

Exercise 3. *Prove the above basic fact.*

Notation 1 (Bracket Notation For Secret Shares)**.** *Suppose our n parties hold XOR shares of some secret $x \in \{0,1\}^\ell$. It is very common to use bracket notation $[\![x]\!]$ to denote the collection of shares. Formally, this is shorthand for an ordered tuple of shares that XOR to x:*

$$[\![x]\!] = (x_0, ..., x_{n-1}) \qquad s.t. \qquad \bigoplus_i x_i = x.$$

It is important to note that this notation does not necessarily imply that the shares are uniform, simply that they XOR to x.

Exercise 4. *Above, we for concreteness consider a secret sharing scheme over bitstrings of length ℓ, but it is also possible to consider secret sharing over other sets. In particular, it is easy to construct an n-out-of-n secret sharing scheme for any finite group G (e.g., the integers modulo some m, invertible square matrices, etc.). Show our above XOR scheme generalizes to any such group and check it satisfies the desired correctness and security properties.*

Operations on Secret Shares. Suppose we have two secret-shared bits $[\![x]\!], [\![y]\!]$ where $x, y \in \{0,1\}$. It will be useful to define an XOR operation over such secret shares. Namely, to XOR $[\![x]\!]$ and $[\![y]\!]$, each party will XOR their local shares:

$$[\![x]\!] \oplus [\![y]\!] = (x_0, ..., x_{n-1}) \oplus (y_0, ..., y_{n-1}) \triangleq (x_0 \oplus y_0, ..., x_{n-1} \oplus y_{n-1}).$$

Crucially, it is not hard to see that the resulting sharing is of form $[\![x \oplus y]\!]$.

Similarly, we can define multiplication by a constant:

$$c \cdot [\![x]\!] = c \cdot (x_0, ..., x_{n-1}) \triangleq (c \cdot x_0, ..., c \cdot x_{n-1}) = [\![c \cdot x]\!].$$

Exercise 5. *Use properties of $\oplus$ to prove that $[\![x]\!] \oplus [\![y]\!] = [\![x \oplus y]\!]$ and that $c \cdot [\![x]\!] = [\![c \cdot x]\!]$.*

It is very common to refer to XOR/multiplication by a constant as homomorphisms. E.g., you might hear that a secret sharing scheme is XOR-homomorphic; this means that performing some operation on shares (i.e., element-wise XOR) translates to some operation on the shared value (i.e., XOR).

Exercise 6 (Harder)**.** *Following the previous exercise, is it possible to define addition/scalar multiplication for arbitrary finite groups? If not, when can we define such operations? Hint: the above operations are called homomorphisms for a reason.*

Computing AND/Correlated Randomness

Now that we have XOR secret shares in hand, we can work on implementing three-party secure AND. Recall that today's considered computation is as follows:

> Parties A, B hold respective input bits $x, y \in \{0,1\}$. Party D has no input. The parties agree to compute the AND $x \cdot y$ and to deliver the result only to parties A and B. That is, D should not obtain output.

We start with a bit of bookkeeping, and a chance to revisit the definition of semi-honest security. We update our definition of semi-honest security to handle more than two parties, as well as computations that output different values to different parties:

Definition 3 (p-Party, One Corruption Semi-Honest Security (for deterministic computations))**.** *Let $f_0, ..., f_{p-1}$ be computable functions. Let $P_0, ..., P_{p-1}$ be parties with respective inputs $x_0, ..., x_{p-1}$. We say that a protocol Π securely computes $f_0, ..., f_{p-1}$ in the presence of a semi-honest adversary corrupting one party when (1) the protocol is **correct** (party P_i outputs $f_i(x0, ..., x_{p-1})$, except with negligible probability) and (2) for each party P_i, there exists a probabilistic polynomial time simulator $\mathcal{S}_i$ such that for all inputs $x_0, ..., x_{p-1}$, the following hold:*

$$\texttt{View}_i^{\Pi}(x_0, ..., x_{p-1}) \stackrel{c}{=} \mathcal{S}_i(x_i, f_i(x_0, ..., x_{p-1}))$$

If the above symbols $\stackrel{c}{=}$ are replaced by $\approx$, we say that Π has statistical security. If the above symbols $\stackrel{c}{=}$ are replaced by $\equiv$, we say that Π has perfect security.

In our above scenario, we consider three parties, where one party named D has no input and has no output, i.e. x_D is empty, often written $\perp$, and the function f_D outputs $\perp$.

Beaver's Trick. As already discussed, it is difficult for A and B to securely multiply their bits alone. However, suppose that A and B have somehow already securely multiplied secret shares of two uniformly random bits.

In particular, suppose that A and B somehow hold the following information:

$$[\![\alpha]\!], [\![\beta]\!], [\![\alpha \cdot \beta]\!] \qquad \text{where} \qquad \alpha, \beta \leftarrow \{0, 1\}.$$

That is, they hold three secret shared bits, α, β, and $\alpha \cdot \beta$. In the literature, such a triple of shared values is often called a multiplication triple, or sometimes a Beaver triple, after Donald Beaver, a pioneer of many techniques in MPC [Bea92]. A multiplication triple is a form of what is sometimes called correlated randomness: our parties hold random strings, but those two strings are correlated with one another. In this case, the correlation is that they form a multiplication triple. Correlated randomness is a central concept in MPC.

Given such a triple, there is a relatively straightforward way for A and B to securely multiply their input bits x and y:

- A calls $[\![x]\!] \leftarrow \texttt{Share}(x)$ to obtain two XOR shares of x. A sends one share to B such that they jointly hold $[\![x]\!]$.
- B similarly arranges that A, B hold $[\![y]\!]$.
- A, B use their ability to XOR secret shares of values to locally compute $[\![x \oplus \alpha]\!]$ and $[\![y \oplus \beta]\!]$.
- A, B reconstruct $x \oplus \alpha$ and $y \oplus \beta$. In particular, A sends her share of $x \oplus \alpha$ (resp. $y \oplus \beta$) to B, and vice versa, then the parties locally call `Reconstruct`. Informally, the reason it is safe to reconstruct these values is because α (resp. β) is uniform, and hence it acts as a one-time pad mask on x (resp. y), perfectly hiding the true value.
- Now that both parties know $x \oplus \alpha, y \oplus \beta$, they can use homomorphisms on secret shares to compute the following:

  $$(x \oplus \alpha) \cdot [\![y]\!] \oplus (y \oplus \beta) \cdot [\![\alpha]\!] \oplus [\![\alpha \cdot \beta]\!].$$

  The claim is that this computes a secret share $[\![x \cdot y]\!]$:

  $$\begin{aligned}
  &(x \oplus \alpha) \cdot [\![y]\!] \oplus (y \oplus \beta) \cdot [\![\alpha]\!] \oplus [\![\alpha \cdot \beta]\!] \\
  &= [\![(x \oplus \alpha) \cdot y \oplus (y \oplus \beta) \cdot \alpha \oplus \alpha \cdot \beta]\!] \\
  &= [\![(x \cdot y \oplus \alpha \cdot y) \oplus (y \cdot \alpha \oplus \beta \cdot \alpha) \oplus \alpha \cdot \beta]\!] \\
  &= [\![x \cdot y \oplus \alpha \cdot y \oplus \alpha \cdot y \oplus \alpha \cdot \beta \oplus \alpha \cdot \beta]\!] \\
  &= [\![x \cdot y]\!].
  \end{aligned}$$

- Now that the parties have $[\![x \cdot y]\!]$, they can simply exchange shares and reconstruct the output $x \cdot y$.

Exercise 7. *Above, the step where A (resp. B) send bits to share x (resp. y) is not needed. Can you see why?*

Now, the only remaining question is: Where does the multiplication triple $[\![\alpha]\!], [\![\beta]\!], [\![\alpha \cdot \beta]\!]$ come from? Clearly, in the three party setting, we can use our party D to sample these shares and to deal them to the two parties.

Let's see a full protocol:

Protocol 2 (Two-Party AND with a Dealer)**.** *The protocol runs between two parties A and B and a dealer D.*

- ***Input:*** *Party A inputs a bit $x \in \{0,1\}$; B inputs a bit $y \in \{0,1\}$. D inputs nothing.*
- ***Output:*** *Parties A and B output $x \cdot y$. D outputs nothing.*
- ***Procedure:***
  - *D flips two coins $\alpha, \beta \leftarrow \{0,1\}$ and computes $\alpha \cdot \beta$.*
  - *D generates 2-out-of-2 XOR secret shares of $\alpha, \beta, \alpha \cdot \beta$:*

    $$[\![\alpha]\!] \leftarrow \texttt{Share}(\alpha) \qquad [\![\beta]\!] \leftarrow \texttt{Share}(\beta) \qquad [\![\alpha \cdot \beta]\!] \leftarrow \texttt{Share}(\alpha \cdot \beta).$$

    D deals one share of each sharing to A and one to B. At this point, D will no longer be needed in the protocol.
  - *From here, the parties follow the steps described above to obtain $x \cdot y$.*

The protocol above is secure in the semi-honest model when no two parties collude:

Theorem 1. *Protocol 2 is perfectly secure against a semi-honest adversary that corrupts exactly one party.*

Proof. By constructing simulators for each of the parties.

Recall that in the semi-honest model, we wish to show that the adversary's view (i.e., the adversary's input, randomness, and received messages) looks similar to the output of some algorithm (a simulator) that runs in an ideal world. It turns out that for this particular protocol, we can obtain perfect security, meaning that our simulators can output information that is identically distributed to the real-world views.

Let's quickly simulate the dealer D. This is trivial: in the real world, the dealer does not receive any messages. So the simulator for D can be constructed by simply running real-world D and writing down the results of its coin flips; this is all D sees in the real world.

Simulating A, B is a bit harder. Let's focus on simulating what A sees; since the protocol is symmetric, so is the proof.

First, recall the rules. Our simulator for A lives in the ideal world, where it gets as input A's input x and the function output $x \cdot y$. From this, it must produce something that looks identical to A's real-world view. So, let's go step-by-step through the information A sees in the real world, and explain what the simulator will do to account for that information. A's view includes:

- Her shares of $\alpha, \beta, \gamma = \alpha \cdot \beta$, received from D. Recall that each of these shares is constructed by a call to `Share`, and that these shares look uniform. Thus, the simulator can simulate each of these three bits by flipping a fair coin.
- Randomness resulting from the call to $\texttt{Share}(x)$. Since the simulator knows x, it can just run `Share` itself and produce identically distributed randomness.
- An XOR share of y received from B. Again, this share is uniform by construction, so the simulator just flips a fair coin.
- B's shares of $x \oplus \alpha$ (resp. $y \oplus \beta$). This share can also be simulated by flipping a fair coin, though the reasoning is slightly different. In particular, the reason we can simulate B's share of $x \oplus \alpha$ with a uniform bit is because α itself is (1) uniform and (2) independent of everything A has seen so far. This means that $x \oplus \alpha$ also appears uniform to A, and hence so does B's share of $x \oplus \alpha$ (the exact same reasoning applies for the received share of $y \oplus \beta$).

- B's share of $x \cdot y$. Let this share be m. Here, we must be careful: the share m sent by B does not look uniform to A. Indeed, prior to receiving m, A has already seen quite a lot of information, and m is correlated with that information. Based on the information A already saw in the real-world, recall that she herself holds a share of $x \cdot y$. Let this share be s. The message m is then precisely what is needed such that A can reconstruct the correct output $x \cdot y$. In particular, the simulator chooses

$$m = (x \cdot y) \oplus s.$$

The above is a full accounting of all information A sees, and our simulator can perfectly simulate all of it. B's simulator is defined similarly.

To be extremely pedantic, we can write A's view and simulator precisely. Note that normally we would be okay with a less pedantic accounting of such details. First, her view:

$$\mathtt{View}_A^{\Pi}(x,y) := \left\{ (x, x_A, y_A, \alpha_A, \beta_A, \gamma_A, x_B \oplus \alpha_B, y_B \oplus \beta_B, m_B) \;\middle|\; \begin{array}{l} \alpha, \beta \leftarrow \{0,1\} \\ \gamma := \alpha \cdot \beta \\ x_A, y_A \leftarrow \{0,1\} \\ \alpha_A, \beta_A, \gamma_A \leftarrow \{0,1\} \\ x_B := x \oplus x_A \\ y_B := y \oplus y_A \\ \alpha_B := \alpha \oplus \alpha_A \\ \beta_B := \beta \oplus \beta_A \\ \gamma_B := \gamma \oplus \gamma_A \\ m_B := (x \oplus \alpha) y_B \oplus (y \oplus \beta) \alpha_B \oplus \gamma_B \end{array} \right\}$$

Above, we precisely account for all random shares seen by A, as well as the three messages sent by B. Note that the view is a function of both parties' inputs; again, it is a mechanical accounting of the information A sees, as a function of both parties' inputs. The most complex term is m_B, which is the ultimate message sent from B that opens the product xy.

The simulator for A is as follows:

$$\mathtt{Sim}_A^{\Pi}(x,xy) := \left\{ (x, x_A, y_A, \alpha_A, \beta_A, \gamma_A, x_B \oplus \alpha_B, y_B \oplus \beta_B, m_B) \;\middle|\; \begin{array}{l} \alpha, \beta \leftarrow \{0,1\} \\ \gamma := \alpha \cdot \beta \\ x_A, y_A \leftarrow \{0,1\} \\ \alpha_A, \beta_A, \gamma_A \leftarrow \{0,1\} \\ x_B := x \oplus x_A \\ y_B := 0 \\ y := y_A \\ \alpha_B := \alpha \oplus \alpha_A \\ \beta_B := \beta \oplus \beta_A \\ \gamma_B := \gamma \oplus \gamma_A \\ m_A := (x \oplus \alpha) y_A \oplus (y \oplus \beta) \alpha_A \oplus \gamma_A \\ m_B := xy \oplus m_A \end{array} \right\}$$

The most complex part of the above simulation is that B's input y is simply syntactically unavailable. Instead, the simulator "makes up" a value for y, being careful to ensure this preserves the output distribution. Notice that m_B is now no longer derived from B's shares, but rather solved for from A's shares and the output. As an exercise, carefully look to ensure the distributions are identical. Namely, for every x, y:

$$\mathtt{View}_A^{\Pi}(x,y) \equiv \mathtt{Sim}_A^{\Pi}(x,xy)$$

Since we have a perfect simulation of each party individually, we have proved our claim. □

Next Time

We will next see how to use what we have learned today to execute any computable function between three parties. Looking forward, we will use our techniques from today to securely execute each of the gates in some Boolean circuit.

We will also soon see how to remove the assumption of a third party/no collusions.

References

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Berlin, Heidelberg, August 1992.