2

Defining Multi-Party Computation

In this chapter, we introduce notations and conventions we will use throughout, define some basic cryptographic primitives, and provide a security definition for multi-party computation. Although we will not focus on formal security proofs or complete formal definitions, it is important to have clear security definitions to understand exactly what properties protocols are designed to provide. The protocols we discuss in later chapters have been proven secure based on these definitions.

2.1 Notations and Conventions

We will abbreviate *Secure Multi-Party Computation* as *MPC*, and will use it to denote secure computation among two or more participants. The term *secure function evaluation* (SFE) is often used to mean the same thing, although it can also apply to contexts where only one party provides inputs to a function that is evaluated by an outsourced server. Because two-party MPC is an important special case, which received a lot of targeted attention, and because two-party protocols are often significantly different from the general n-party case, we will use 2PC to emphasize this setting when needed.

We assume existence of direct secure channels between each pairs of

participating players. Such channels could be achieved inexpensively through a variety of means, and are out of scope in this book.

We denote encryption and decryption of a message m under key k as $\mathsf{Enc}_k(m)$ and $\mathsf{Dec}_k(m)$. We will refer to protocol participants interchangeably also as parties or players, and will usually denote them as P_1, P_2, etc. We will denote the adversary by $\mathcal{A}$.

A negligible function $\nu : \mathbb{N} \to \mathbb{R}$ is any function that approaches zero asymptotically faster than any inverse polynomial. In other words, for any polynomial p, $\nu(n) < 1/p(n)$ for all but finitely many n.

We will denote computational and statistical security parameters by κ and σ respectively. The computational security parameter κ governs the hardness of problems that can be broken by an adversary's offline computation — e.g., breaking an encryption scheme. In practice κ is typically set to a value like 128 or 256. Even when we consider security against computationally bounded adversaries, there may be some attacks against an interactive protocol that are not made easier by offline computation. For example, the interactive nature of a protocol may give the adversary only a single opportunity to violate security (e.g., by sending a message that has a special property, like predicting the random value that an honest party will chose in the next round). The statistical security parameter σ governs the hardness of these attacks. In practice, σ is typically set to a smaller value like 40 or 80. The correct way to interpret the presence of two security parameters is that security is violated only with probability $2^{-\sigma} + \nu(\kappa)$, where ν is a negligible function *that depends on the resources of the adversary.* When we consider computationally unbounded adversaries, we omit κ and require $\nu = 0$.

We will use symbol $\in_R$ to denote uniformly random sampling from a distribution. For example we write "choose $k \in_R \{0,1\}^\kappa$" to mean that k is a uniformly chosen κ-bit long string. More generally, we write "$v \in_R D$" to denote sampling according to a probability distribution D. Often the distribution in question is the output of a randomized algorithm. We write "$v \in_R A(x)$" to denote that v is the result of running randomized algorithm A on input x.

Let D_1 and D_2 be two probability distributions indexed by a security parameter, or equivalently two algorithms that each take a security parameter as input.[1] We say that D_1 and D_2 are *indistinguishable* if, for all algorithms A

[1] In the literature, D_1 and D_2 are often referred to as an *ensemble* of distributions.

there exists a negligible function ν such that:

$$\Pr[A(D_1(n)) = 1] - \Pr[A(D_2(n)) = 1] \leq \nu(n)$$

In other words, no algorithm behaves more than negligibly differently when given inputs sampled according to D_1 vs D_2. When we consider only *non-uniform, polynomial-time* algorithms A, the definition results in *computational indistinguishability.* When we consider *all* algorithms without regard to their computational complexity, we get a definition of *statistical indistinguishability.* In that case, the probability above is bounded by the *statistical distance* (also known as total variation distance) of the two distributions, which is defined as:

$$\Delta(D_1(n), D_2(n)) = \frac{1}{2} \sum_x \left| \Pr[x = D_1(n)] - \Pr[x = D_2(n)] \right|$$

Throughout this work, we use *computational security* to refer to security against adversaries implemented by non-uniform, polynomial-time algorithms. We use *information-theoretic security* (also known as *unconditional* or *statistical* security) to mean security against arbitrary adversaries (even those with unbounded computational resources).

2.2 Basic Primitives

Here, we provide definitions of a few basic primitives we use in our presentation. Several other useful primitives are actually special cases of MPC (i.e., they are defined as MPC of specific functions). These are defined in Section 2.4.

Secret Sharing. Secret sharing is another essential primitive, which is at the core of many MPC approaches. Informally, a (t, n)-secret sharing scheme splits the secret s into n shares, such that any $t - 1$ of the shares reveal no information about s, while any t shares allow complete reconstruction of the secret s. There are many variants of possible security properties of secret sharing schemes; we provide one definition, adapted from Beimel and Chor (1993), next.

Definition 2.1. Let D be the domain of secrets and D_1 be the domain of shares. Let $\mathsf{Shr} : D \to D_1^n$ be a (possibly randomized) sharing algorithm, and $\mathsf{Rec} : D_1^k \to D$ be a reconstruction algorithm. A (t, n)-secret sharing scheme is a pair of algorithms $(\mathsf{Shr}, \mathsf{Rec})$ that satisfies these two properties:

- *Correctness*. Let $(s_1, s_2, ..., s_n) = \mathsf{Shr}(s)$. Then,

$$\Pr[\forall k \geq t, Rec(s_{i_1}, ...s_{i_k}) = s] = 1.$$

- *Perfect Privacy*. Any set of shares of size less than t does not reveal anything about the secret in the information theoretic sense. More formally, for any two secrets $a, b \in D$ and any possible vector of shares $\boldsymbol{v} = v_1, v_2, ..., v_k$, such that $k < t$,

$$\Pr[\boldsymbol{v} = \mathsf{Shr}(a)|_k] = \Pr[\boldsymbol{v} = \mathsf{Shr}(b)|_k],$$

  where $|_k$ denotes appropriate projection on a subspace of k elements.

In many of our discussions we will use (n, n)-secret sharing schemes, where all n shares are necessary and sufficient to reconstruct the secret.

Random Oracle. Random Oracle (RO) is a heuristic model for the security of hash functions, introduced by Bellare and Rogaway (1993). The idea is to treat the hash function as a public, idealized random function. In the random oracle model, all parties have access to the public function $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$, implemented as a stateful oracle. On input string $x \in \{0,1\}^*$, H looks up its history of calls. If $H(x)$ had never been called, H chooses a random $r_x \in \{0,1\}^\kappa$, remembers the pair x, r_x and returns r_x. If $H(x)$ had been called before, H returns r_x. In this way, the oracle realizes a randomly-chosen function $\{0,1\}^* \rightarrow \{0,1\}^\kappa$.

The random oracle model is a heuristic model, because it captures only those attacks that treat the hash function H as a black-box. It deviates from reality in that it models a public function (e.g., a standardized hash function like SHA-256) as an inherently random object. In fact, it is possible to construct (extremely contrived) schemes that are secure in the random oracle model, but which are insecure whenever H is instantiated by *any* concrete function (Canetti *et al.*, 1998).

Despite these shortcomings, the random oracle model is often considered acceptable for practical applications. Assuming a random oracle often leads to significantly more efficient constructions. In this work we will be careful to state when a technique relies on the random oracle model.

2.3 Security of Multi-Party Computation

Informally, the goal of MPC is for a group of participants to learn the correct output of some agreed-upon function applied to their private inputs without revealing anything else. We now provide a more formal definition to clarify the security properties MPC aims to provide. First, we present the *real-ideal paradigm* which forms the conceptual core of defining security. Then we discuss two different adversary models commonly used for MPC. Finally, we discuss issues of composition—namely, whether security preserved in the natural way when a secure protocol invokes another subprotocol.

2.3.1 Real-Ideal Paradigm

A natural way to define security is to come up with a kind of a "laundry list" of things that constitute a violation of security. For example, the adversary should not be able to learn a certain predicate of another party's input, the adversary should not be able to induce impossible outputs for the honest parties, and the adversary should not be able to make its inputs depend on honest parties' inputs. Not only is this a tedious approach, but it is cumbersome and error-prone. It is not obvious when the laundry list could be considered complete.

The real-ideal paradigm avoids this pitfall completely by introducing an incredibly clear "ideal world" that implicitly captures all security guarantees, and defining security in relation to this ideal world. The definition of probabilistic encryption by Goldwasser and Micali (1984) is widely considered to be the first instance of using this approach to define and prove security, although they used different terminology.

Ideal World. In the ideal world, the parties securely compute the function $\mathcal{F}$ by privately sending their inputs to a completely trusted party $\mathcal{T}$, referred to as the *functionality*. Each party P_i has an associated input x_i, which is sends to $\mathcal{T}$, which simply computes $\mathcal{F}(x_1, \ldots, x_n)$ and returns the result to all parties. Often we will make a distinction between $\mathcal{F}$ as a trusted party (functionality) and the circuit C that such a party computes on the private inputs.

We can imagine an adversary attempting to attack the ideal-world interaction. An adversary can take control over any of the parties P_i, but not $\mathcal{T}$ (that is the sense in which $\mathcal{T}$ is described as a *trusted* party). The simplicity

of the ideal world makes it easy to understand the effect of such an attack. Considering our previous laundry list: the adversary clearly learns no more than $\mathcal{F}(x_1, \ldots, x_n)$ since that is the only message it receives; the outputs given to the honest parties are all consistent and legal; the adversary's choice of inputs is independent of the honest parties'.

Although the ideal world is easy to understand, the presence of a fully-trusted third party makes it imaginary. We use the ideal world as a benchmark against which to judge the security of an actual protocol.

Real World. In the real world, there is no trusted party. Instead, all parties communicate with each other using a protocol. The protocol π specifies for each party P_i a "next-message" function π_i. This function takes as input a security parameter, the party's private input x_i, a random tape, and the list of messages P_i has received so far. Then, π_i outputs either a next message to send along with its destination, or else instructs the party to terminate with some specific output.

In the real world, an adversary can corrupt parties—corruption at the beginning of the protocol is equivalent to the original party being an adversary. Depending on the threat model (discussed next), corrupt parties may either follow the protocol as specified, or deviate arbitrarily in their behavior.

Intuitively speaking, the real world protocol π is considered secure if any effect that an adversary can achieve in the real world can also be achieved by a corresponding adversary in the ideal world. Put differently, the goal of a protocol is to provide security in the real world (given a set of assumptions) that is equivalent to that in the ideal world.

2.3.2 Semi-Honest Security

A *semi-honest* adversary is one who corrupts parties but follows the protocol as specified. In other words, the corrupt parties run the protocol honestly but they may try to learn as much as possible from the messages they receive from other parties. Note that this may involve several colluding corrupt parties pooling their views together in order to learn information. Semi-honest adversaries are also considered *passive* in that they cannot take any actions other than attempting to learn private information by observing a view of a protocol execution. Semi-honest adversaries are also commonly called *honest-but-curious*.

The *view* of a party consists of its private input, its random tape, and the list of all messages received during the protocol. The view of an adversary consists of the combined views of all corrupt parties. Anything an adversary learns from running the protocol must be an efficiently computable function of its view. That is, without loss of generality we need only consider an "attack" in which the adversary simply outputs its entire view.

Following the real-ideal paradigm, security means that such an "attack" can also be carried out in the ideal world. That is, for a protocol to be secure, it must be possible in the ideal world to generate something indistinguishable from the real world adversary's view. Note that the adversary's view in the ideal world consists of nothing but inputs sent to $\mathcal{T}$ and outputs received from $\mathcal{T}$. So, an ideal-world adversary must be able to use this information to generate what looks like a real-world view. We refer to such an ideal-world adversary as a *simulator*, since it generates a "simulated" real-world view while in the ideal-world itself. Showing that such a simulator exists proves that there is nothing an adversary can accomplish in the real world that could not also be done in the ideal world.

More formally, let π be a protocol and $\mathcal{F}$ be a functionality. Let C be the set of parties that are corrupted, and let Sim denote a simulator algorithm. We define the following distributions of random variables:

- $\mathsf{Real}_\pi(\kappa, C; x_1, \ldots, x_n)$: run the protocol with security parameter κ, where each party P_i runs the protocol honestly using private input x_i. Let V_i denote the final view of party P_i, and let y_i denote the final output of party P_i.
  Output $\{V_i \mid i \in C\}, (y_1, \ldots, y_n)$.

- $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(\kappa, C; x_1, \ldots, x_n)$: Compute $(y_1, \ldots, y_n) \leftarrow \mathcal{F}(x_1, \ldots, x_n)$.
  Output $\mathsf{Sim}(C, \{(x_i, y_i) \mid i \in C\}), (y_1, \ldots, y_n)$.

A protocol is secure against semi-honest adversaries if the corrupted parties in the real world have views that are indistinguishable from their views in the ideal world:

Definition 2.2. A protocol π *securely realizes* $\mathcal{F}$ *in the presence of semi-honest adversaries* if there exists a simulator Sim such that, for every subset of corrupt parties C and all inputs $x_1, \ldots, x_n$, the distributions

$$\mathsf{Real}_\pi(\kappa, C; x_1, \ldots, x_n)$$

and

$$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(\kappa, C; x_1, \ldots, x_n)$$

are indistinguishable (in κ).

In defining Real and Ideal we have included the outputs of all parties, even the honest ones. This is a way of incorporating a correctness condition into the definition. In the case that no parties are corrupt ($C = \emptyset$), the output of Real and Ideal simply consist of all parties' outputs in the two interactions. Hence, the security definition implies that protocol gives outputs which are distributed just as their outputs from the ideal functionality (and this is true even when $\mathcal{F}$ is randomized). Because the distribution of $y_1, \ldots, y_n$ in Real does not depend on the set C of corrupted parties (no matter who is corrupted, the parties all run honestly), it is not strictly necessary to include these values in the case of $C \neq \emptyset$, but we choose to include it to have a unified definition.

The semi-honest adversary model may at first glance seem exceedingly weak—simply reading and analyzing received messages barely even seems like an attack at all! It is reasonable to ask why such a restrictive adversary model is worth considering at all. In fact, achieving semi-honest security is far from trivial and, importantly, semi-honest protocols often serve as a basis for protocols in more robust settings with powerful attackers. Additionally, many realistic scenarios do correspond to semi-honest attack behavior. One such example is computing with players who are trusted to act honestly, but cannot fully guarantee that their storage might not be compromised in the future.

2.3.3 Malicious Security

A *malicious* (also known as *active*) may instead cause corrupted parties to deviate arbitrarily from the prescribed protocol in an attempt to violate security. A malicious adversary has all the powers of a semi-honest one in analyzing the protocol execution, but may also take any actions it wants during protocol execution. Note that this subsumes an adversary that can control, manipulate, and arbitrarily inject messages on the network (even through throughout this book we assume direct secure channels between each pair of parties). As before, security in this setting is defined in comparison to the ideal world, but there are two important additions to consider:

Effect on honest outputs. When the corrupt parties deviate from the protocol, there is now the possibility that honest parties' outputs will be affected. For example, imagine an adversary that causes two honest parties to output different things while in the ideal world all parties get identical outputs. This condition is somewhat trivialized in the previous definition — while the definition does compare real-world outputs to ideal-world outputs, these outputs have no dependence on the adversary (set of corrupted parties). Furthermore, we can/should make no guarantees on the final outputs of corrupt parties, only of the honest parties, since a malicious party can output whatever it likes.

Extraction. Honest parties follow the protocol according to a well-defined input, which can be given to $\mathcal{T}$ in the ideal world as well. In contrast, the input of a malicious party is not well-defined in the real world, which leads to the question of what input should be given to $\mathcal{T}$ in the ideal world. Intuitively, in a secure protocol, whatever an adversary can do in the real world should also be achievable in the ideal world by *some suitable choice of inputs* for the corrupt parties. Hence, we leave it to the simulator to choose inputs for the corrupt parties. This aspect of simulation is called *extraction*, since the simulator extracts an effective ideal-world input from the real-world adversary that "explains" the input's real-world effect. In most constructions, it is sufficient to consider *black-box* simulation, where the simulator is given access only to the oracle implementing the real-world adversary, and not its code.

When $\mathcal{A}$ denotes the adversary program, we write $\mathsf{corrupt}(\mathcal{A})$ to denote the set of parties that are corrupted, and use $\mathsf{corrupt}(\mathsf{Sim})$ for the set of parties that are corrupted by the ideal adversary, Sim. As we did for the semi-honest security definition, we define distributions for the real world and ideal world, and define a secure protocol as one that makes those distributions indistinguishable:

- $\mathsf{Real}_{\pi,\mathcal{A}}(\kappa; \{x_i \mid i \notin \mathsf{corrupt}(\mathcal{A})\})$: run the protocol on security parameter κ, where each honest party P_i (for $i \notin \mathsf{corrupt}(\mathcal{A})$) runs the protocol honestly using given private input x_i, and the messages of corrupt parties are chosen according to $\mathcal{A}$ (thinking of $\mathcal{A}$ as a protocol next-message function for a collection of parties). Let y_i denote the output of each

honest party P_i and let V_i denote the final view of party P_i.
Output $(\{V_i \mid i \in \mathsf{corrupt}(\mathcal{A})\}, \{y_i \mid i \notin \mathsf{corrupt}(\mathcal{A})\})$.

- $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(\kappa; \{x_i \mid i \notin \mathsf{corrupt}(\mathcal{A})\})$: Run Sim until it outputs a set of inputs $\{x_i \mid i \in \mathsf{corrupt}(\mathcal{A})\}$. Compute $(y_1, \ldots, y_n) \leftarrow \mathcal{F}(x_1, \ldots, x_n)$. Then, give $\{y_i \mid i \in \mathsf{corrupt}(\mathcal{A})\}$ to Sim.[2] Let V^* denote the final output of Sim (a set of simulated views).
Output $(V^*, \{y_i \mid i \notin \mathsf{corrupt}(\mathsf{Sim})\})$.

Definition 2.3. A protocol π *securely realizes* $\mathcal{F}$ *in the presence of malicious adversaries* if for every real-world adversary $\mathcal{A}$ there exists a simulator Sim with $\mathsf{corrupt}(\mathcal{A}) = \mathsf{corrupt}(\mathsf{Sim})$ such that, for all inputs for honest parties $\{x_i \mid i \notin \mathsf{corrupt}(\mathcal{A})\}$, the distributions

$$\mathsf{Real}_{\pi,\mathcal{A}}(\kappa; \{x_i \mid i \notin \mathsf{corrupt}(\mathcal{A})\})$$

and

$$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(\kappa; \{x_i \mid i \notin \mathsf{corrupt}(\mathsf{Sim})\})$$

are indistinguishable (in κ).

Note that the definition quantifies only over the inputs of honest parties $\{x_i \mid i \notin \mathsf{corrupt}(\mathcal{A})\}$. The interaction Real does not consider the corrupt parties to have any inputs, and the inputs of the corrupt parties in Sim is only determined indirectly (by the simulator's choice of what to send to $\mathcal{F}$ on the corrupt parties' behalf). While it would be possible to also define inputs for corrupt parties in the real world, such inputs would merely be "suggestions" since corrupt parties could choose to run the protocol on any other input (or behave in a way that is inconsistent with all inputs).

Reactive functionalities. In the ideal world, the interaction with the functionality consists of just a single round: inputs followed by outputs. It is possible to generalize the behavior of $\mathcal{F}$ so that it interacts with the parties over many rounds of interaction, keeping its own private internal state between rounds. Such functionalities are called *reactive*.

[2] To be more formal, we can write the simulator Sim as a pair of algorithms $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ which capture this two-phase process. Sim_1 (on input κ) outputs $\{x_i \mid i \in \mathsf{corrupt}(\mathcal{A})\}$ and arbitrary internal state Σ. Then Sim_2 takes input Σ and $\{y_i \mid i \in \mathsf{corrupt}(\mathcal{A})\}$, and gives output V^*.

One example of a reactive functionality is as the dealer in a poker game. The functionality must keep track of the state of all cards, taking input commands and giving outputs to all parties in many rounds.

Another example is an extremely common functionality called *commitment*. This functionality accepts a bit b (or more generally, a string) from P_1 and gives output "committed" to P_2, while internally remembering b. At some later time, if P_1 sends the command "reveal" (or "open") to the functionality, it gives b to P_2.

Security with abort. In any message-based two-party protocol, one party will learn the final output before the other. If that party is corrupt and malicious, they may simply refuse to send the last message to the honest party and thereby prevent the honest party from learning the output. However, this behavior is incompatible with our previous description of the ideal world. In the ideal world, if corrupt parties receive output from the functionality then all parties do. This property is called *output fairness* and not all functions can be computed with this property (Cleve, 1986; Gordon *et al.*, 2008; Asharov *et al.*, 2015a).

Typical results in the malicious setting provide a weaker property known as *security with abort*, which requires slightly modifying the ideal functionality as follows. First, the functionality is allowed to know the identities of the corrupt parties. The functionality's behavior is modified to be slightly reactive: after all parties have provided input, the functionality computes outputs and delivers the outputs to the *corrupt parties only*. Then the functionality awaits either a "deliver" or "abort" command from the corrupted parties. Upon receiving "deliver", the functionality delivers the outputs to all the honest parties. Upon receiving "abort", the functionality delivers an abort output ($\perp$) to all the honest parties.

In this modified ideal world, an adversary is allowed to learn the output before the honest parties and to prevent the honest parties from receiving any output. It is important to note, however, that whether an honest party aborts can depend only on the corrupt party's outputs. In particular, it would violate security if the honest party's abort probability to depended on its own input.

Usually the possibility of blocking outputs to honest parties is not written explicitly in the description of the functionality. Instead, it is generally understood that when discussing security against malicious adversaries, the

adversary has control over output delivery to honest parties and output fairness is not expected.

Adaptive corruption. We have defined both the real and ideal worlds so that the identities of the corrupted parties are fixed throughout the entire interaction. This provides what is known as security against *static corruption*. It is also possible to consider scenarios where an adversary may choose which parties to corrupt during the protocol execution, possibly based on what it learns during the interaction. This behavior is known as *adaptive corruption*.

Security against adaptive corruption can be modeled in the real-ideal paradigm, by allowing the adversary to issue a command of the form "corrupt P_i". In the real world, this results in the adversary learning the current view (including private randomness) of P_i and subsequently taking over control of its protocol messages. In the ideal world, the simulator learns only the input and outputs of the party upon corruption, and must use this information to generate simulated views. Of course, the views of parties are correlated (if P_i sends a message to P_j, then that message is included in both parties' views). The challenge of adaptive security is that the simulator must produce views piece-by-piece. For example, the simulator may be asked to produce a view of P_i when that party is corrupted. Any messages sent by P_j to P_i must be simulated without knowledge of P_j's private input. Later, the simulator might be asked to provide a view of P_j (including its private randomness) that "explains" its protocol messages as somehow consistent with whatever private input it had.

In this work we consider only static corruption, following the vast majority of work in this field.

2.3.4 Hybrid Worlds and Composition

In the interest of modularity, it is often helpful to design protocols that make use of other ideal functionalities. For example, we may design a protocol π that securely realizes some functionality $\mathcal{F}$, where the parties of π also interact with another functionality $\mathcal{G}$ in addition to sending messages to each other. Hence, the real world for this protocol includes $\mathcal{G}$, while the ideal world (as usual) includes only $\mathcal{F}$. We call this modified real world the *$\mathcal{G}$-hybrid world*.

A natural requirement for a security model is *composition*: if π is a $\mathcal{G}$-hybrid protocol that securely realizes $\mathcal{F}$ (i.e., parties in π send messages and also interact with an ideal $\mathcal{G}$), and ρ is a protocol that securely realizes $\mathcal{G}$, then composing π and ρ in the natural way (replacing every invocation of $\mathcal{G}$ with a suitable invocation of ρ) also results in a secure protocol for $\mathcal{F}$. While we have not defined all of the low-level details of a security model for MPC, it may be surprising that some very natural ways of specifying the details *do not guarantee composability* of secure protocols!

The standard way of achieving guaranteed composition is to use the *universal composability* (UC) framework from Canetti (2001). The UC framework augments the security model that we have sketched here with an additional entity called the *environment*, which is included in both the ideal and real worlds. The purpose of the environment is to capture the "context" in which the protocol executes (e.g., the protocol under consideration is invoked as a small step in some larger calling protocol). The environment chooses inputs for the honest party and receives their outputs. It also may interact arbitrarily with the adversary.

The same environment is included in the real and ideal worlds, and its "goal" is to determine whether it is instantiated in the real or ideal world. Previously we defined security by requiring certain real and ideal views to be indistinguishable. In this setting, we can also absorb any distinguisher of these views into the environment itself. Hence, without loss of generality, the environment's final output can be just a single bit which can be interpreted as the environment's "guess" of whether it is instantiated in the real or ideal world.

Next, we define the real and ideal executions, where Z is an environment:

- $\mathsf{Real}_{\pi,\mathcal{A},Z}(\kappa)$: run an interaction involving adversary $\mathcal{A}$ and environment Z. When Z generates an input for an honest party, the honest party runs protocol π, and gives its output to Z. Finally, Z outputs a single bit, which is taken as the output of $\mathsf{Real}_{\pi,\mathcal{A},Z}(\kappa)$.

- $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},Z}(\kappa)$: run an interaction involving adversary (simulator) Sim and environment Z. When Z generates an input for an honest party, the input is passed directly to functionality $\mathcal{F}$ and the corresponding output is given to Z (on behalf of that honest party). The output bit of Z is taken as the output of $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},Z}(\kappa)$.

Definition 2.4. A protocol π *UC-securely realizes* $\mathcal{F}$ if for all real-world adversaries $\mathcal{A}$ there exists a simulator Sim with $\mathsf{corrupt}(\mathcal{A}) = \mathsf{corrupt}(\mathsf{Sim})$ such that, for all environments Z:

$$\left|\Pr[\mathsf{Real}_{\pi,\mathcal{A},Z}(\kappa) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},Z}(\kappa) = 1]\right| \text{ is negligible (in } \kappa).$$

Since the definition quantifies over all environments, we can always consider absorbing the adversary $\mathcal{A}$ into the environment Z, so that what is left over is the so-called "dummy adversary" (which simply forwards protocol messages as instructed by Z).

In other (non UC-composable) security models, the ideal-world adversary (simulator) can depend arbitrarily on the real-world adversary. In particular, the simulator can do things like internally run the adversary and repeatedly rewind that adversary to a previous internal state. Many protocols are proven in these weaker model where the composability may be restricted. Sequential composition security (i.e., security for protocols which call functionalities in a sequential manner) holds for all protocols discussed in this book.

In the UC model such rewinding is not possible since the adversary can be assumed to be absorbed into the environment, and the simulator is not allowed to depend on the environment. Rather, the simulator must be a *straight-line simulator*: whenever the environment wishes to send a protocol message, the simulator must reply immediately with a simulated response. A straight-line simulator must generate the simulated transcript in one pass, whereas the previous definitions allowed for the simulated transcript or view to be generated without any restrictions. Assuming the other primitives (e.g., OT and commitments) used in these protocols provide UC-security, the malicious secure protocols described in this book are all UC-secure.

2.4 Specific Functionalities of Interest

Here, we define several functionalities that have been identified as particularly useful building blocks for building MPC protocols.

Oblivious Transfer. Oblivious Transfer (OT) is an essential building block for secure computation protocols. It is theoretically equivalent to MPC as shown by Kilian (1988): given OT, one can build MPC without any additional assumptions, and, similarly, one can directly obtain OT from MPC.

PARAMETERS:

1. Two parties: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$. $\mathcal{S}$ has input secrets $x_0, x_1 \in \{0,1\}^n$, and $\mathcal{R}$ has a selection bit $b \in \{0,1\}$.

FUNCTIONALITY:

- $\mathcal{R}$ receives x_b, $\mathcal{S}$ receives $\perp$.

Figure 2.1: 1-out-of-2 OT functionality $\mathcal{F}^{\text{OT}}$.

PARAMETERS:

1. Two parties: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$. Length of committed string n.

FUNCTIONALITY:

- $\mathcal{S}$ sends a string $s \in \{0,1\}^n$ to $\mathcal{F}^{\text{Comm}}$. $\mathcal{F}^{\text{Comm}}$ sends **committed** to $\mathcal{R}$.
- At some later time, $\mathcal{S}$ sends **open** to $\mathcal{F}^{\text{Comm}}$. $\mathcal{F}^{\text{Comm}}$ sends s to $\mathcal{R}$.

Figure 2.2: Commitment functionality $\mathcal{F}^{\text{Comm}}$.

The standard definition of 1-out-of-2 OT involves two parties, a Sender $\mathcal{S}$ holding two secrets x_0, x_1 and a receiver $\mathcal{R}$ holding a choice bit $b \in \{0,1\}$. OT is a protocol allowing $\mathcal{R}$ to obtain x_b while learning nothing about the "other" secret x_{1-b}. At the same time, $\mathcal{S}$ does not learn anything at all. More formally:

Definition 2.5. A 1-out-of-2 OT is a cryptographic protocol securely implementing the functionality $\mathcal{F}^{\text{OT}}$ of Figure 2.1.

Later in this chapter we define more formally what it means for a cryptographic protocol to be *secure* in this setting.

Many variants of OT may be considered. A natural variant is 1-out-of-k OT, in which $\mathcal{S}$ holds k secrets, and $\mathcal{R}$ has a choice selector from $[0, \ldots, k-1]$. We discuss protocols for implementing OT efficiently in Section 3.7.

Commitment. Commitment is a fundamental primitive in many cryptographic protocols. A commitment scheme allows a sender to commit to a

PARAMETERS:

1. Two parties: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$.

FUNCTIONALITY:

- $\mathcal{P}$ sends (C, x) to $\mathcal{F}^{\text{zk}}$, where $C : \{0,1\}^n \to \{0,1\}$ is a Boolean circuit with 1 output bit, and $x \in \{0,1\}^n$. If $C(x) = 1$ then $\mathcal{F}^{\text{zk}}$ sends (**proven**, C) to $\mathcal{V}$. Otherwise, it sends $\perp$ to $\mathcal{V}$.

Figure 2.3: Zero-knowledge proof functionality $\mathcal{F}^{\text{zk}}$.

secret value, and reveal it at some later time to a receiver. The receiver should learn nothing about the committed value before it is revealed by the sender (a property refered to as *hiding*), while the sender should not be able to change its choice of value after committing (the *binding* property).

Commitment is rather simple and inexpensive in the random oracle model. To commit to x, simply choose a random value $r \in_R \{0,1\}^\kappa$ and publish the value $y = H(x\|r)$. To later reveal, simply announce x and r.

Definition 2.6. *Commitment* is a cryptographic protocol implementing the functionality $\mathcal{F}^{\text{Comm}}$ of Figure 2.2.

Zero-Knowledge Proof. A zero-knowledge (ZK) proof allows a prover to convince a verifier that it knows x such that $C(x) = 1$, without revealing any further information about x. Here C is a public predicate.

As a simple example, suppose G is a graph and that Alice knows a 3-coloring χ for G. Then Alice can use a ZK proof to convince Bob that G is 3-colorable. She constructs a circuit C_G that interprets its input as an encoding of a 3-coloring and checks whether it is a legal 3-coloring of G. She uses (C_G, χ) as input to the ZK proof. From Bob's point of view, he receives output (**proven**, C_G) if and only if Alice was able to provide a valid 3-coloring of G. At the same time, Alice knows that Bob learned nothing about her 3-coloring χ other than the fact that *some* legal χ exists.

Definition 2.7. A *zero-knowledge proof* is a cryptographic protocol implementing the functionality $\mathcal{F}^{\text{zk}}$ of Figure 2.3.

There are several variants of ZK proofs identified in the literature. Our specific variant is more precisely a zero-knowledge *argument of knowledge*. The distinctions between these variants are not crucial for the level of detail we explore in this book.

2.5 Further Reading

The real-ideal paradigm was first applied in the setting of MPC by Goldwasser *et al.* (1985), for the special case of zero-knowledge. Shortly thereafter the definition was generalized to arbitrary MPC by Goldreich *et al.* (1987). These definitions contain the important features of the real-ideal paradigm, but resulted in a notion of security (against malicious adversaries) that was not preserved under composition. In other words, a protocol could be secure according to these models *when executed in isolation*, but may be totally insecure when two protocol instances are run concurrently.

The definition of security that we have sketched in this book is the Universal Composition (UC) framework of Canetti (2001). Protocols proven secure in the UC framework have the important composition property described in Section 2.3.4, which in particular guarantees security of a protocol instance no matter what other protocols are executing concurrently. While the UC framework is the most popular model with this property, there are other models with similar guarantees (Pfitzmann and Waidner, 2000; Hofheinz and Shoup, 2011). The details of all such security models are extensive and subtle. However, a significantly simpler model is presented by Canetti *et al.* (2015), which is equivalent to the full UC model for the vast majority of cases. Some of the protocols we describe are secure in the random oracle model. Canetti *et al.* (2014) describe how to incorporate random oracles into the UC framework.

Our focus in this book is on the most popular security notions — namely, semi-honest security and malicious security. The literature contains many variations on these security models, and some are a natural fit for real-world applications. We discuss some alternative security models in Chapter 7.