- *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 $H : \{0,1\}^* \to \{0,1\}^\kappa$, 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 $\{0,1\}^* \to \{0,1\}^\kappa$.

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 "ideal world" that implicitly captures all security guarantees, and defining security in relation to this ideal world. Although they used different terminology, 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.

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