Chapter 13

Secure Multiparty Computation

Kannan Balasubramanian
Mepco Schlenk Engineering College, India

M. Rajakani
Mepco Schlenk Engineering College, India

ABSTRACT

The Secure Multiparty computation is characterized by computation by a set of multiple parties each participating using the private input they have. There are different types of models for Secure Multiparty computation based on assumption about the type of adversaries each model is assumed to protect against including Malicious and Covert Adversaries. The model may also assume a trusted setup with either using a Public Key Infrastructure or a using a Common Reference String. Secure Multiparty Computation has a number of applications including Scientific Computation, Database Querying and Data Mining.

INTRODUCTION

Secure Multiparty Computation (MPC) involves carrying out computation tasks by a set of parties based on the private inputs they each party has. Research in the MPC area has focused on only a limited set of problems. These computations could occur between mutually untrusted parties, or even between competitors. For example, customers might send to a remote database queries that contain private information or two competing financial organizations might jointly invest in a project that must satisfy both organizations’ private constraints. To conduct such computations, one entity must usually know the inputs from all the participants; however, if nobody can be trusted enough to know all the inputs, privacy will become a primary concern. We present models for the Secure multiparty computation problem and provide examples for them. We also provide a classification of the Secure multiparty computation problems based on the type of adversaries against them.

DOI: 10.4018/978-1-5225-2915-6.ch013
**SECURE MULTIPARTY COMPUTATION**

In general, a secure multi-party computation problem deals with computing any probabilistic function on any input, in a distributed network where each participant holds one of the inputs, ensuring independence of the inputs, correctness of the computation, and that no more information is revealed to a participant in the computation than can be inferred from that participant’s input and output. A common strategy is to assume the trustworthiness of the service providers, or to assume the existence of a trusted third party which is risky in today’s dynamic and malicious environment.

Examples of secure multiparty computation problems include privacy-preserving database querying, scientific computations, intrusion detection, statistical analysis, geometric computations, and data mining. It is possible to systematically transform normal computations (not necessarily security related) to secure multi-party computations. The common property of the above three problems is the following: two or more parties want to conduct a computation based on their private inputs, but neither party is willing to disclose its own input to anybody else. The problem is how to conduct such a computation while preserving the privacy of the inputs. This problem is referred to as Secure Multi-party Computation problem or MPC (Yao, 1982).

**A MODEL FOR SECURE MULTIPARTY COMPUTATION**

Secure multi-party computation (MPC) can be defined as the problem of n players computing an agreed function of their inputs in a secure way, where security means guaranteeing the correctness of the output as well as the privacy of the players’ inputs, even when some players cheat. Concretely, we assume we have inputs $x_1, ..., x_n$, where player $i$ knows $x_i$, and we want to compute $f(x_1, ..., x_n) = (y_1, ..., y_n)$ such that player $i$ is guaranteed to learn $y_i$, but can get nothing more than that.

An example is the Yao’s millionaire’s problem: two millionaires meet in the street and want to find out who is richer. Can they do this without having to reveal how many millions they each own? The function computed in this case is a simple comparison between two integers. If the result is that the first millionaire is richer, then he knows that the other guy has fewer millions than him, but this should be all the information he learns about the other guy’s fortune.

Another example is a voting scheme: here all players have an integer as input, designating the candidate they vote for, and the goal is to compute how many votes each candidate has received. We want to make sure that the correct result of the vote, but only this result, is made public. In these examples, all players learn the same result, i.e., $y_1 = ... = y_n$, but it can also be useful to have different results for different players. Consider for example the case of a blind signature scheme, which is useful in electronic cash systems. We can think of this as a two-party secure computation where the signer enters his private signing key $sk$ as input, the user enters a message $m$ to be signed, and the function $f(sk, m) = (y_1, y_2)$, where $y_1$ is for the signer and is empty, and where $y_2$ is for the user and the signature on $m$. Again, security means exactly what we want: the user gets the signature and nothing else, while the signer learns nothing new.

It is clear that if we can compute any function securely, we have a very powerful tool. However, some protocol problems require even more general ways of thinking. A secure payment system, for instance, cannot naturally be formulated as secure computation of a single function: what we want here is to continuously keep track of how much money each player has available and avoid cases where for instance people spend more money than they have. Such a system should behave like a secure general-purpose