Chapter 12
Secure Two Party Computation

Kannan Balasubramanian
Mepco Schlenk Engineering College, India

ABSTRACT

The goal of secure two-party computation is to enable two parties to cooperatively evaluate a function that takes private data from both parties as input without exposing any of the private data. At the end of the computation, the participants learn nothing more than the output of the function. The two-party secure computation systems have three properties: (1) the application involves inputs from two independent parties; (2) each party wants to keep its own data secret; and (3) the participants agree to reveal the output of the computation. That is, the result itself does not imply too much information about either party’s private input. Informally, the security requirements are that nothing is learned from the protocol other than the output (privacy), and that the output is distributed according to the prescribed functionality (correctness). The threat models in the two-party computation assume the presence of three different types of adversaries: 1) Semi honest, 2) Malicious and 3) Covert.

INTRODUCTION

The goal of secure two-party computation is to enable two parties to cooperatively evaluate a function that takes private data from both parties as input without exposing any of the private data. At the end of the computation, the participants learn nothing more than the output of the function. Secure computation has many important applications such as privacy-preserving biometric identification, set intersection and finding the $k$th ranked element. Practical solutions to the two-party computation are scarce due to the high runtime costs associated with traditional techniques and the effort required to build them. In this chapter, we review the threat models for the two-party computation and solutions for securing the two-party computation for each of the threat models.

MODEL OF TWO PARTY COMPUTATION

The two-party secure computation systems have three properties: (1) the application involves inputs from two independent parties; (2) each party wants to keep its own data secret; and (3) the participants agree
Secure Two Party Computation

to reveal the output of the computation. That is, the result itself does not imply too much information about either party’s private input.

More formally, in the setting of two-party computation, two parties with respective private inputs x and y, wish to jointly compute a functionality \( f(x, y) = (f_1(x, y), f_2(x, y)) \), such that the first party receives \( f_1(x, y) \) and the second party receives \( f_2(x, y) \). This functionality may be probabilistic, in which case \( f(x, y) \) is a random variable. Informally, the security requirements are that nothing is learned from the protocol other than the output (privacy), and that the output is distributed according to the prescribed functionality (correctness).

THREAT MODELS

The threat models in the two-party computation assume the presence of three different types of adversaries (Huang, 2012): 1) Semi honest, 2) Malicious and 3) Covert.

The semi-honest (also known as honest-but-curious) threat model, assume that all parties follow the protocol as specified, but may attempt to learn additional information about the other party’s input from the protocol transcript. It is usually unrealistic to assume passive adversaries who always obey the protocol specifications. To compromise the protocol security, an active adversary can deviate from the protocol in arbitrary ways, even at the risk of being caught. Informally speaking, a two-party computation protocol is said to be secure in the malicious threat model if the privacy and correctness properties are guaranteed even in presence of such active adversaries. In the covert model, a cheating adversary is “caught” with some constant probability, but with the remaining probability can (potentially) learn the honest party’s entire input and arbitrarily bias the honest party’s output. If an adversary is unwilling to take the risk of being caught, then such protocols will deter cheating altogether.

The first general solution for the problem of secure two-party computation in the presence of semi-honest adversaries was presented by (Yao, 1986). Later, solutions were provided for the multi-party and malicious adversarial cases by Goldreich et al. (Goldreich et al., 1987).

YAO’S PROTOCOL

Let \( f \) be a polynomial-time functionality (assume for now that it is deterministic), and let \( x \) and \( y \) be the parties’ respective inputs. The first step is to view the function as a Boolean circuit \( C \). In order to describe Yao’s protocol, it is helpful to first recall how such a circuit is computed. Let \( x \) and \( y \) be the parties’ inputs. Then, the circuit \( C(x, y) \) is computed gate-by-gate, from the input wires to the output wires. Once the incoming wires to a gate \( g \) have obtained values \( \alpha, \beta \in \{0,1\} \), it is possible to give the outgoing wires of the gate the value \( g(\alpha ; \beta) \). The output of the circuit is given by the values obtained in the output wires of the circuit. Thus, essentially, computing a circuit involves allocating appropriate zero-one values to the wires of the circuit. In the description below, we refer to four different types of wires in a circuit: circuit-input wires (that receive the input values \( x \) and \( y \)), circuit-output wires (that carry the value \( C(x, y) \)), gate-input wires (that enter some gate \( g \)), and gate-output wires (that leave some gate \( g \)).

We now present a high-level description of Yao’s protocol. The construction is actually a “compiler” that takes any polynomial-time functionality \( f \), or actually a circuit \( C \) that computes \( f \), and constructs a protocol for securely computing \( f \) in the presence of semi-honest adversaries. In a secure protocol, the