Secure Computation for Privacy Preserving Data Mining

Yehuda Lindell
Bar-Ilan University, Israel

INTRODUCTION

The increasing use of data mining tools in both the public and private sectors raises concerns regarding the potentially sensitive nature of much of the data being mined. The utility to be gained from widespread data mining seems to come into direct conflict with an individual's need and right to privacy. Privacy preserving data mining solutions achieve the somewhat paradoxical property of enabling a data mining algorithm to use data *without* ever actually "seeing" it. Thus, the benefits of data mining can be enjoyed, without compromising the privacy of concerned individuals.

BACKGROUND

A classical example of a privacy preserving data mining problem is from the field of medical research. Consider the case that a number of different hospitals wish to jointly mine their patient data, for the purpose of medical research. Furthermore, let us assume that privacy policy and law prevents these hospitals from ever pooling their data or revealing it to each other due to the confidentiality of patient records. In such a case, classical data mining solutions cannot be used. Fortunately, privacy preserving data mining solutions enable the hospitals to compute the desired data mining algorithm on the union of their databases, without ever pooling or revealing their data. Indeed, the only information (provably) learned by the different hospitals is the output of the data mining algorithm. This problem whereby different organizations cannot directly share or pool their databases, but must nevertheless carry out joint research via data mining, is quite common. For example, consider the interaction between different intelligence agencies in the USA. These agencies are suspicious of each other and do not freely share their data. Nevertheless, due to recent security needs, these agencies must run data mining algorithms on their combined data. Another example relates to data that is held by governments. Until recently, the Canadian Government held a vast federal database that pooled citizen data from a number of different government ministries (this database was called the "big brother" database by some). The Canadian government claimed that the database was essential for research. However, due to privacy concerns and public outcry, the database was dismantled, thereby preventing that "essential research" from being carried out. This is another example of where privacy preserving data mining could be used to balance between real privacy concerns and the need of governments to carry out important research.

Privacy preserving data mining is actually a special case of a long-studied problem in cryptography: *secure multiparty computation*. This problem deals with a setting where a set of parties with private inputs wish to jointly compute some function of their inputs. Loosely speaking, this joint computation should have the property that the parties learn the correct output and nothing else, even if some of the parties maliciously collude to obtain more information.

MAIN THRUST

In this short chapter, we will provide a succinct overview of secure multiparty computation, and how it can be applied to the problem of privacy preserving data mining. Our main focus will be on how security is formally defined, why this definitional approach is adopted, and what issues should be considered when defining security for privacy preserving data mining problems. Due to space constraints, the treatment in this chapter is both brief and informal. For more details, we refer the reader to (Goldreich, 2003) for a survey on cryptography and cryptographic protocols.

Security Definitions for Secure Computation

The aim of a secure multiparty computation task is for the participating parties to *securely* compute some function of their distributed and private inputs. However, what does it mean for a computation to be *secure*? One way of approaching this question is to provide a list of *security properties* that should be preserved. The first such property that often comes to mind is that of *privacy* or *confidentiality*. A naïve attempt at formalizing privacy would be to require that each party learns nothing about the other parties' inputs, even if it behaves maliciously. However, such a definition is usually unattainable because the defined output of the computation itself typically reveals some information on other parties' inputs. (For example, a decision tree computed on two distributed databases reveals some information about both databases.) Therefore, the privacy requirement is usually formalized by saying that the only information learned by the parties in the computation (again, even by those who behave maliciously) is that specified by the function output. Although privacy is a primary security property, it rarely suffices. Another important property is that of *correctness*; this states the honest parties' outputs are correctly distributed even in the face of adversarial attack. A central question that arises in this process of defining security properties is: *when is our list of properties complete*? This question is, of course, application-dependent and this essentially means that for every new problem, the process of deciding which security properties are required must be re-evaluated. We stress that coming up with the right list of properties is often very difficult, and it can take many years until we are convinced that a definition truly captures the security requirements that are needed. Furthermore, an incomplete of properties may easily lead to real security failures.

The Ideal/Real Model Paradigm

Due to these difficulties, the standard definitions of secure computation today follow an alternative approach called the *ideal/real model paradigm*. This has been the dominant paradigm in the investigation of secure computation in the last fifteen years; we refer the reader to (Canetti, 2000) for the formal definition and references therein for related definitional work. Loosely speaking, this paradigm defines the security of a real protocol by comparing it to an *ideal computing scenario* in which the parties interact with an external trusted and incorruptible party. In this ideal execution, the parties all send their inputs to the trusted party (via ideally secure communication lines). The trusted party then computes the function on these inputs and sends each party its specified output. Such a computation embodies the goal of secure computation, and it is easy to see that the properties of privacy and correctness hold in the ideal model. In addition to the fact that these and other security properties are preserved in an ideal execution, the simplicity of the ideal model provides an intuitively convincing security guarantee. For example, notice that the only message that a party sends in an ideal execution is its input, and so the only power that a corrupted party has is to choose its input (something which is typically legitimate anyway).

So far, we have defined an ideal execution in an ideal world. However, in the *real world*, the parties run a protocol without any trusted help. Despite this, a secure real protocol should somehow "emulate" an ideal execution. That is, we say that a real protocol that is run by the parties (in a world where no trusted party exists) is ***secure***, if *no adversary* can do more harm in

Box 1.