Part I

Secure Multiparty Computation

1 Introduction

1.1 Private Information, Uses and Misuses

In a modern information-driven society, the everyday life of individuals and companies is full of cases where various kinds of private information are important resources. While a cryptographer might think of PIN codes and keys in this context, these types of secrets are not our main concern here. Rather, we will talk about information that is closer to the *primary business* of an individual or a company. For a private person, this may be data concerning his or her economic situation, such as income, loans, and tax data, or information about his or her health, such as diseases and medicine usage. For a company, it might be the customer database or information on how the business is running, such as turnover, profits, and salaries.

What is a viable strategy for handling private information? Finding a good answer to this question has become more complicated in recent years. When computers were in their infancy, in the 1950s and 1960s, electronic information security was to a large extent a military business. A military organization is quite special in that confidential information needs to be communicated almost exclusively between its own members, and the primary security goal is to protect this communication from being leaked to external enemies. While it may not be trivial to reach this goal, at least the overall purpose is simple to state and understand.

In modern society, things get much more complicated: using electronic media, we need to interact and do business with a large number of parties, some of whom we have never met and many of whom may have interests that conflict with ours. So how do you handle your confidential data if you cannot be sure that the parties you interact with are trustworthy?

One could save the sensitive data in a very secure location and never access it, but this is, of course, unreasonable. Our private data usually have value only because we want to use them for something. In other words, we have to have ways of controlling leakage of confidential data while these data are being stored, communicated, or computed on, *even in cases where the owner of the data does not trust the parties he or she communicates with.*

A very interesting fact that makes this problem even more important is that there are many scenarios in which a large amount of added value can be obtained by *combining confidential information from several sources* and from this computing some result that holds an interest for all parties. To illustrate what we mean by this, we look at a number of different example scenarios in the following subsections.

1.1.1 Auctions

Auctions exist in many variants and are used for all kinds of purposes, but we concentrate here on the simple variant where some item is for sale and the highest bid wins. We assume that the auction is conducted in the usual way, where the price starts at some preset amount and people place increasing bids until no one wants to bid more than the currently highest bid. When you enter such an auction, you usually have some (more or less precisely defined) idea of the maximum amount you are willing to pay and therefore when you will stop bidding. However, every bidder, of course, wants to pay as small a price as possible for the item. Indeed, the winner of the auction may hope to pay less than his or her maximum amount. This will happen if all other bidders stop participating long before the current bid reaches this maximum.

For such an auction to work in a fair way, it is obvious that the maximum amount you are willing to pay should be kept private. For instance, if the auctioneer knows your maximum and is working with another bidder, they can force the price to be always just below your maximum and so force you to pay more than if the auction had been honest. Note that the auctioneer has an incentive to do this to increase his or her own income, which is often a percentage of the price the item is sold for. However, the result of the auction could, in principle, be computed if one were given as input the true maximum value each bidder assigns to the item on sale.

1.1.2 Procurement

A procurement system is a sort of inverted auction, where some party (typically a public institution) asks companies to bid for a contract, that is, to make an offer on the price for doing a certain job. In such a case, the lowest bid usually wins. However, bidders are typically interested in getting as high a price as possible.

It is obvious that bids are private information: a participating company is clearly not interested in competitors learning its bid before it has to make its own bid. This would allow the competitors to beat the company's bid by always offering a price that is slightly lower than that offered by the company. This is also against the interests of the institution offering the contract because it will tend to make the winning bid higher. The result of the process, namely, who wins the contract, can, in principle, be computed given all the true values of the bids.

1.1.3 Benchmarking

Assume that you run a company. You will naturally be interested in how well you are doing compared with other companies in the same line of business as yours. The comparison may be concerned with a number of different parameters, such as profit relative to size, average salaries, and productivity. Other companies will most likely have similar interests in such a comparison, which is known as a *benchmark analysis*. Such an analysis takes input from all participating companies. Based on this, it tries to compute information on how well a company in the given line of business should be able to perform, and each company is told how its performance compares with this "ideal."

It is clear that each company will insist that its own data are private and must not be leaked to competitors. However, the desired results can be computed from the private data: there are several known methods from information economics for doing such an analysis efficiently.

1.1.4 Data Mining

In most countries, public institutions such as the tax authorities or the health care system keep databases containing information on citizens. In many cases, there are advantages one can get from coordinated access to several such databases. Researchers may be able to get statistics they could not get otherwise, or institutions might get an administrative advantage from being able to quickly gather the information they need on a certain individual.

However, there is clearly a privacy concern here: access to many different databases by a single party opens the possibility that complete dossiers could be compiled on particular citizens, which would be a violation of privacy. In fact, accessing data on the same person in several distinct databases is forbidden by law in some countries precisely because of this concern.

1.2 Do We Have to Trust Someone?

We are now in a position to extract some common features of all the scenarios we have looked at so far. One way to describe them all is as follows: we have a number of parties, and each possesses some private data. We want to do some computation that needs all the private data as input. The parties are interested in learning the result, or at least some part of it, but want to keep their private data as confidential as possible.

Hopefully, it is clear from the preceding section that if we can find a satisfactory solution to this problem, a very large number of applications would benefit. Moreover, this leads to an extremely intriguing theoretical question, as we now explain:

One possible – and trivial – solution would be to find some party T that everyone is willing to trust. All parties privately give their input to T, who does the required computation, announces the result to the parties, and forgets about the private data he or she has seen. A moment's thought will show that this is hardly a satisfactory solution: first, we have created a single point of attack from which all the private data can potentially be stolen. Second, the parties must all completely trust T with respect to both privacy and correctness of the results. The reason why there are privacy concerns is that the parties do not trust each other in the first place, so why should we believe that they can find a new party they all trust?

In some applications, one may pay a party T for doing the computation; if the amount paid is thought to be larger than what T could gain from cheating, the parties may be satisfied with the solution. This seems to work in some cases, for instance, when a consultancy house is paid a large fee for doing a benchmark analysis – but this is, of course, a very expensive solution.

Thus, we are left with a fundamental question: *can the problem be solved without relying on a trusted party?*

At first sight, it may seem that this cannot be possible. We want to compute a result that depends on private data from *all* involved parties. How could one possibly do this unless data from several parties become known to someone, and hence we have to trust that party?

Nevertheless, as we shall see, the problem is by no means impossible to solve, and solutions do exist that are satisfactory, both from theoretical and from practical points of view.

1.3 Multiparty Computation

Let us be slightly more precise about the problem we have to solve: the parties, or *players*, that participate are called $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Each player P_i holds a secret input x_i, and the players agree on some function f that takes n inputs. Their goal is to compute $y = f(x_1, \ldots, x_n)$ while making sure that the following two conditions are satisfied:

- Correctness: the correct value of y is computed; and
- Privacy: y is the *only* new information that is released.

Regarding the latter property, note that because the purpose of the whole exercise is to learn y, the best we can hope for in terms of privacy is that nothing but y is leaked. Computing f such that privacy and correctness are achieved is referred to as computing f *securely*. Later in this book we will be precise about what secure computing is; for now, we will be content with the preceding intuitive idea. Note also that one may consider a more general case where each player gets his or her own private output. We will do so later; for now, we focus on a single, public output for simplicity.

As one example of how this connects to the scenarios from the preceding section, one may think of x_i as a number, namely, P_i's bid in an auction, and $f(x_1, ..., x_n) = (z, j)$, where $x_j = z$ and $z \geq x_i, i = 1, \ldots, n$; that is, f outputs the highest bid and the identity of the corresponding bidder. If we do not want the winner to pay his or her own bid but the bid of the second-highest bidder, we simply change z to be this value, which is again a well-defined function of the inputs. This would give us a function implementing a so-called second price auction.

In this section we give a first intuition on how one might compute a function securely without relying on trusted parties. This requires that we specify a *protocol*, that is, a set of instructions that players are supposed to follow to obtain the desired result. For simplicity, we will assume for now that players always follow the protocol. We will later address the case in which some parties may deviate from the protocol in order to get more information than they are supposed to or cause the result to be incorrect. We will also assume that any pair of players can communicate securely; that is, it is possible for P_i to send a message m to P_j such that no third party sees m, and P_j knows that m came from P_i. We discuss later how this can be realized in practice.

1.3.1 Secure Addition and Voting

Let us first look at a simple special case, namely, where each x_i is a natural number and $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$. Secure computation of even such a simple function can have very meaningful applications. Consider the case where $\mathsf{P}_1, ..., \mathsf{P}_n$ want to vote on some yes/no decision. Then we can let x_i represent the vote of P_i, where $x_i = 0$ means "no" and $x_i = 1$

means "yes." If we can compute the sum of the x_i securely, this exactly means that we get a way to vote with the properties we usually expect: the result $\sum_{i=1}^n x_i$ is indeed the result of the vote, namely, the number of yes votes. Moreover, if the computation is secure, no information is leaked other than $\sum_{i=1}^n x_i$; in particular, no information is revealed on how a particular player voted.

We will now design a protocol for the voting application. To be consistent with the next example, we set $n = 3$. A later exercise shows how to construct a voting solution for any n.

Secret Sharing

Before we can solve the problem, we need to look at an important tool known as *secret sharing*. The term may seem self-contradictory at first sight: how can anything be secret if you share it with others? Nevertheless, the name makes good sense: the point is that secret sharing provides a way for a party, say P_1, to spread information on a secret number x across all the players such that they together hold full information on x, yet no player (except, of course, P_1) has any information on x. First, we choose a prime p, and we define $\mathbb{Z}_p$ as $\mathbb{Z}_p = \{0, 1, ..., p-1\}$.[1] In the following, we will think of the secret x as a number in $\mathbb{Z}_p$.

In order to *share the secret s*, P_1 chooses numbers r_1, r_2 uniformly at random in $\mathbb{Z}_p$ and sets

$$r_3 = x - r_1 - r_2 \bmod p$$

Put another way, P_1 chooses r_1, r_2, r_3 randomly from $\mathbb{Z}_p$, subject to the constraint that $x = r_1 + r_2 + r_3 \bmod p$. Note that this way of choosing r_1, r_2, r_3 means that each of the three numbers is uniformly chosen in $\mathbb{Z}_p$: for each of them, all values in $\mathbb{Z}_p$ are possible and equally likely. Now P_1 sends privately r_1, r_3 to P_2, r_1, r_2 to P_3, and keeps r_2, r_3 himself or herself. The r_js are called the *shares* of the *secret x*.

The process we have described satisfies two basic properties: First, the secret x is kept private in the sense that neither P_2 nor P_3 knows anything about that secret. As a result, if some hacker breaks into the machine of P_2 or P_3 (but not both), he or she will learn nothing about x. Second, x can be reconstructed if shares from at least two players are available. Let's argue that this is true in a more precise way:

Privacy. Even though P_1 has distributed shares of the secret x to the other players, neither P_2 nor P_3 has any idea what x is. For P_2, we can argue as follows: he or she knows r_1, r_3 (but not r_2) and that $x = r_1 + r_2 + r_3 \bmod p$. Take any $x_0 \in \mathbb{Z}_p$. From P_2's point of view, could it be that $x = x_0$? The answer is yes, for if $x = x_0$, it would have to be the case that $r_2 = x_0 - r_1 - r_3 \bmod p$. This is certainly a possibility. Recall that r_2 is uniformly chosen in $\mathbb{Z}_p$, so all values are possible. However, any other choice, say $x = x_0' \neq x_0$, is also a possibility. If this were the answer, we would have $r_2 = x_0' - r_1 - r_3 \bmod p$, which is a value that is different from $x_0 - r_1 - r_3 \bmod p$ but just as likely. We conclude that what has been sent to P_2 reveals nothing new about x. A similar argument shows that the same is true from P_3's point of view.

Correctness. If two of the three parties pool their information, the secret can be reconstructed because then all three shares will be known, and one can simply add them modulo p.

[1] To be more precise, $\mathbb{Z}_p$ is another name for $\mathbb{Z}/p\mathbb{Z}$, where we identify $i \in \mathbb{Z}_p$ with the residue class of numbers that are congruent to i modulo p. See more details in Chapter 2.

Note that the privacy property is *information theoretic*: as long as a party does not know all three summands, no amount of computing power can give that party any new information on the corresponding secret. In this book, we focus primarily on protocols with this type of security. The secret-sharing technique just shown is a special case of so-called replicated secret sharing. There are many ways to realize secret sharing with other desirable properties than the method we show here, and we look at several such techniques later, as well as a more general definition of what secret sharing is.

A Protocol for Secure Addition

The basic idea for secure addition is that all players $\mathsf{P}_1, \mathsf{P}_2$, and P_3 will distribute shares of their private values x_1, x_2, and x_3 in exactly the way we saw before. It turns out that one can now compute the sum securely by locally adding shares and announcing the result. The complete protocol is as follows:

Protocol Secure Addition

Participants are $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$; input for P_i is $x_i \in \mathbb{Z}_p$, where p is a fixed prime agreed on in advance.

1. Each P_i computes and distributes shares of his or her secret x_i as described in the text: he or she chooses $r_{i,1}, r_{i,2}$ uniformly at random in $\mathbb{Z}_p$ and sets $r_{i,3} = x_i - r_{i,1} - r_{i,2} \bmod p$.
2. Each P_i sends privately $r_{i,2}, r_{i,3}$ to P_1, $r_{i,1}, r_{i,3}$ to P_2, and $r_{i,1}, r_{i,2}$ to P_3 (note that this involves P_i sending "to himself or herself"). So P_1, for instance, now holds $r_{1,2}, r_{1,3}$, $r_{2,2}, r_{2,3}$, and $r_{3,2}, r_{3,3}$.
3. Each P_j adds corresponding shares of the three secrets – more precisely, he or she computes, for $\ell \neq j$, $s_\ell = r_{1,\ell} + r_{2,\ell} + r_{3,\ell} \bmod p$ and announces s_ℓ to all parties. Each party computes and announces two values.
4. All parties compute the result $v = s_1 + s_2 + s_3 \bmod p$.

To analyze the secure addition protocol, let us first see why the result v is indeed the correct result. This is straightforward:

$$v = \sum_j s_j \bmod p = \sum_j \sum_i r_{i,j} \bmod p = \sum_i \sum_j r_{i,j} \bmod p = \sum_i x_i \bmod p$$

This shows that the protocol computes the sum modulo p of the inputs, no matter how the x_i are chosen. However, if we let the parties choose $x_i = 1$ for "yes" and $x_i = 0$ for "no" and make sure that $p > 3$, then $\sum_i x_i \bmod p = \sum_i x_i$ because all x_i are 0 or 1, and so, their sum cannot be larger than p. So, in this case, v is indeed the number of yes votes.

Now why is it the case that no new information other than the result v is leaked to any player? Let us concentrate on P_1 for concreteness. In step 1, x_1, x_2, and x_3 are secrets shared, and we have already argued that this tells P_1 nothing whatsoever about x_2, x_3. In the final step, s_1, s_2, s_3 are announced. Note that P_1 already knows s_2, s_3, so s_1 is the only new piece of information. However, we can argue that seeing s_1 will tell P_1 what v is and nothing more. The reason for this is that if one is given s_2, s_3, and v, one can compute $s_1 = v - s_2 - s_3 \bmod p$. Put another way, given what P_1 is supposed to know, namely, v, we can already compute what he or she sees in the protocol, namely, s_1, and therefore, seeing the information from the protocol tells him or her nothing beyond v.

This type of reasoning is formalized later in this book and is called a *simulation argument*: given what a player is supposed to know, we show how to efficiently compute (simulate) everything he or she sees in the protocol, and from this, we conclude that the protocol tells the player nothing beyond what we wanted to tell him or her.

Note that given the result, P_1 is in fact able to compute some information about other people's votes. In particular, he or she can compute $v - x_1 = x_2 + x_3$, that is, the sum of the other players' votes. It is easy to get confused and think that because of this, something must be wrong with the protocol, but in fact, there is no problem: it is true that P_1 can compute the sum of the votes of P_2 and P_3, but this follows from information P_1 is *supposed to know*, namely, the result and his or her own input. There is nothing the protocol can do to deprive P_1 of such information – in other words, the best a protocol can do is to make sure that players only learn what they are supposed to learn, and this includes whatever can be derived from the player's own input and the intended result.

1.3.2 Secure Multiplication and Matchmaking

To do general secure computation, we will, of course, need to do more than secure addition. It turns out that the secret-sharing scheme from the preceding subsection already allows us to do more: we can also do secure multiplication.

Suppose that two numbers $a, b \in \mathbb{Z}_p$ have been secret shared as described earlier, so that $a = a_1 + a_2 + a_3 \bmod p$ and $b = b_1 + b_2 + b_3 \bmod p$, and we wish to compute the product $ab \bmod p$ securely. We obviously have

$$ab = a_1b_1 + a_1b_2 + a_1b_3 + a_2b_1 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2 + a_3b_3 \bmod p$$

It is now easy to see that if the a_is and b_is have been distributed as described earlier, it is the case that for each product a_ib_j, there is at least one player among the three who knows a_i and b_j and therefore can compute a_ib_j. For instance, P_1 has been given a_2, a_3, b_2, b_3 and can therefore compute a_2b_2, a_2b_3, a_3b_2, and a_3b_3. The situation is therefore that the desired result ab is the sum of some numbers where each summand can be computed by at least one of the players. But now we are essentially done because from Protocol Secure Addition we already know how to add securely!

The protocol resulting from these observations follows. To argue why it works, one first notes that correctness, namely, $ab = u_1 + u_2 + u_3 \bmod p$, follows trivially from the preceding. To show that nothing except $ab \bmod p$ is revealed, one notes that nothing new about a, b is revealed in the first step, and because Protocol Secure Addition is private, nothing except the sum of the inputs is revealed in the last step, and this sum always equals $ab \bmod p$.

It is interesting to note that even in a very simple case where both a and b are either 0 or 1, secure multiplication has a meaningful application: consider two parties, Alice and Bob. Suppose that Alice is wondering whether Bob wants to go out with her, and Bob is asking himself if Alice is interested in him. They would very much like to find out if there is mutual interest but without running the risk of the embarrassment that would result if, for instance, Bob tells Alice that he is interested, only to have Alice turn him down. The problem can be solved if we let Alice choose $a \in \mathbb{Z}_p$, where $a = 1$ if she is interested in Bob and $a = 0$ otherwise. In the same way, Bob chooses b to be 0 or 1. Then we compute the function $f(a, b) = ab \bmod p$ securely. It is clear that the result is 1 if and only

Protocol Secure Multiplication

Participants are $\mathsf{P}_1, \mathsf{P}_2$, and P_3; input for P_1 is $a \in \mathbb{Z}_p$; input for P_2 is $b \in \mathbb{Z}_p$, where p is a fixed prime agreed on in advance. P_3 has no input.

1. P_1 distributes shares a_1, a_2, a_3 of a, while P_2 distributes shares b_1, b_2, b_3 of b.
2. P_1 locally computes $u_1 = a_2b_2 + a_2b_3 + a_3b_2 \bmod p$, P_2 computes $u_2 = a_3b_3 + a_1b_3 + a_3b_1 \bmod p$, and P_3 computes $u_3 = a_1b_1 + a_1b_2 + a_2b_1 \bmod p$.
3. The players use Protocol Secure Addition to compute the sum $u_1 + u_2 + u_3 \bmod p$ securely, where P_i uses u_i as input.

if there is mutual interest. However, if, for instance, Alice is not interested, she will choose $a = 0$, and in this case, she learns *nothing new* from the protocol. To see why, notice that security of the protocol implies that the only (possibly) new information Alice will learn is the result $ab \bmod p$. But she already knows that the result will be 0! In particular, she does not learn whether Bob was interested or not, so Bob is safe from embarrassment. By a symmetric argument, this is, of course, also the case for Alice.

This argument assumes, of course, that both players choose their inputs honestly according to their real interests. In the following section we discuss what happens if players do not follow the instructions and what we can do about the problems resulting from this.

From Protocol Secure Multiplication, we see that if Alice and Bob play the roles of P_1 and P_2, respectively, they just need to find a third party to help them to do the multiplication securely. Note that this third party is not a *completely trusted* third party of the kind we discussed earlier: he or she does not learn anything about a or b other than $ab \bmod p$. Alice and Bob do have to trust, however, that the third party does not share his or her information with Bob or with Alice.

It is an obvious question whether one can do secure multiplication such that *only* Alice and Bob have to be involved? The answer turns out to be yes, but then information-theoretic security is not possible, as we shall see. Instead, one has to use solutions based on cryptography. Such solutions can always be broken if one party has enough computing power, but this is an issue with virtually all the cryptographic techniques used in practice.

For completeness, we remark that Alice and Bob's problem is a special case of the so-called matchmaking problem that has somewhat more serious applications than secure dating. Consider a set of companies where each company has a set of other companies it would prefer to do business with. We want each pair of companies to find out whether there is mutual interest, but without forcing companies to reveal their strategy by announcing their interests in public.

EXERCISE 1.1 Consider the third party helping Alice and Bob to do secure multiplication. Show that the Protocol Secure Multiplication is indeed insecure if the third party reveals what he sees in the protocol to Alice or Bob.

EXERCISE 1.2 We have used replicated secret sharing, where each player receives two numbers in $\mathbb{Z}_p$, even though only one secret number is shared. This was done so that we would be able to do both secure addition and secure multiplication, but for secure addition only, something simpler can be done. Use the principle of writing the secret as a sum of random numbers to design a secret-sharing scheme for any number of parties, where each