The Journal of Privacy and Confidentiality (2009) **1**, Number 1, pp. 59–98

Secure Multiparty Computation for Privacy-Preserving Data Mining

Yehuda Lindell* and Benny Pinkas†

Abstract. In this paper, we survey the basic paradigms and notions of secure multiparty computation and discuss their relevance to the field of privacy-preserving data mining. In addition to reviewing definitions and constructions for secure multiparty computation, we discuss the issue of efficiency and demonstrate the difficulties involved in constructing highly efficient protocols. We also present common errors that are prevalent in the literature when secure multiparty computation techniques are applied to privacy-preserving data mining. Finally, we discuss the relationship between secure multiparty computation and privacy-preserving data mining, and show which problems it solves and which problems it does not.

1 Introduction

Background. Privacy-preserving data mining considers the problem of running data mining algorithms on confidential data that is not supposed to be revealed—even to the party running the algorithm. There are two classic settings for privacy-preserving data mining (although these are by no means the only ones). In the first, the data is divided among two or more different parties; the aim being to run a data mining algorithm on the union of the parties' databases without allowing any party to view another individual's private data. In the second, some statistical data that is to be released (so that it can be used for research using statistics and/or data mining) may contain confidential data; hence, it is first modified so that (a) the data does not compromise anyone's privacy, and (b) it is still possible to obtain meaningful results by running data mining algorithms on the modified data set. In this paper, we will mainly refer to scenarios of the first type.

A classical example of a privacy-preserving data mining problem of the first type occurs in the field of medical research. Consider the case of a number of different hospitals that 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 cases, classical data mining solutions cannot be used. Rather, it is necessary to find a solution that enables the hospitals to compute the desired data mining algorithm on the union of their databases, without ever pooling or revealing their data. Privacy-preserving data mining solutions have the property that 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, yet must nevertheless carry out joint research via data mining, is quite common. For example, consider the interaction between different intelligence agencies. For security purposes, these agencies cannot allow each other free access to their confidential information; if they did, then a single mole in a single agency would have access to an overwhelming number of sources. Nevertheless, as we all know, homeland security also

*Department of Computer Science, Bar-Ilan University, Israel, mailto:lindell@cs.biu.ac.il
†Department of Computer Science, University of Haifa, Israel, mailto:benny@pinkas.net

mandates the sharing of information! It is much more likely that suspicious behavior would be detected if the different agencies were able to run data mining algorithms on their combined data. Another example relates to data that is held by governments. In the late 1990s, the Canadian Government held a vast federal database that pooled citizen data from a number of different government ministries; this database was officially called the Longitudinal Labor Force File, but became known as the "big brother" database. The aim of the database was to implement governmental research that would arguably improve the services received by citizens. However, due to privacy concerns and public outcry, the database was dismantled, thereby preventing such "essential research" from being carried out (46). This is another example where privacy-preserving data mining could be used to balance real privacy concerns with the needs of governments to carry out important research.

Secure computation and privacy-preserving data mining. There are two distinct problems that arise in the setting of privacy-preserving data mining. The first is deciding which functions can be safely computed ("safely" meaning that the privacy of individuals is preserved). For example, is it safe to compute a decision tree on confidential medical data in a hospital, and publicize the resulting tree? This question is not the focus of this paper, but will be discussed briefly in Section 5. For the most part, we will assume that the result of the data mining algorithm is either safe or deemed essential. Thus, the question becomes how to compute the results while minimizing the damage to privacy. For example, it is always possible to pool all of the data in one place and run the data mining algorithm on the pooled data. However, this is exactly what we don't want to do (hospitals are not allowed to hand their raw data out, security agencies cannot afford the risk, and governments risk citizen outcry if they do). Thus, the question we address is how to compute the results without pooling the data in a way that reveals nothing but the final results of the data mining computation.

This question of privacy-preserving data mining is actually a special case of a long-studied problem in cryptography called *secure multiparty computation*. This problem deals with a setting where a set of parties with private inputs wishes 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. Clearly, a protocol that provides this guarantee can be used to solve privacy-preserving data mining problems of the type discussed above.

This paper. In this paper, we present a tutorial-like introduction to secure multiparty computation and discuss its applicability to privacy-preserving data mining. In Section 2 we begin with a light introduction to the model of secure computation, how security is defined, and why. This is followed by full definitions of security in a number of different models. This formal basis is crucial when designing cryptographic protocols for any task, particularly privacy-preserving data mining. Experience shows that cryptographic protocols are extraordinarily hard to get right, and rigorous proofs of security are essential for avoiding subtle flaws that can result in breaches of privacy. We stress that once an individual's privacy is breached, there is no way that the clock can be turned back. Thus, it is not possible to follow a hit-and-miss strategy, whereby a heuristic protocol is deployed and then later replaced if it is deemed not "secure enough". For this reason, we strongly advocate a rigorous approach to this problem. We stress that if it is not possible to construct protocols that are efficient enough in practice and that meet the given definitions of security, then one should search for a different definition of security that is more relaxed (yet one should not give up on the goal of having a rigorous proof of

security relative to some definition). In Section 2.2 we give some examples of how this can be achieved.

Having laid the definitional foundations of secure computation, in Section 3 we proceed to describe the basic tools and paradigms used in constructing secure protocols. Since this paper relates to privacy-preserving data mining, we focus on techniques for constructing highly efficient protocols, thus presenting the best techniques known to date for achieving high efficiency. Given the tutorial-like nature of this paper, in Section 4 we also present some of the common errors that can be found in the literature on privacy-preserving data mining. It is our hope that this will help those new to the field see the subtleties and difficulties that arise when constructing protocols and proving their security.

By its nature, privacy-preserving data mining is a multidisciplinary field. As such, it is our strong belief that it requires close cooperation between researchers and practitioners from the fields of cryptography, data mining, public policy and law. Specifically, most cryptographers are not familiar enough with how data mining really works (knowledge that is clearly necessary for constructing helpful solutions), most data miners are not familiar enough with the subtleties of cryptography and secure computation (making it difficult to construct rigorous cryptographic protocols), and computer science researchers in general are often not familiar enough with the real privacy needs of society. It is our hope that this paper will make the cryptography side of privacy-preserving data mining more accessible to others, thereby contributing to a common language that can be used by researchers from different fields.

We remark that for those readers just interested in understanding the basic notions of secure multiparty computation, but with no interest in constructing protocols, it suffices to read Section 2. Despite this, we suggest that such readers also read Section 3 because a deeper understanding is obtained by seeing how secure multiparty computation protocols are actually constructed. Section 4 is of importance to those who wish to actually construct protocols and can be skipped by others.

Further reading. Although much of this paper can be read with very little background in cryptography, we do assume familiarity with basic concepts such as "computational indistinguishability" when we present formal definitions. An excellent survey by Goldreich (19) provides all of the background necessary for reading this and other advanced papers. For those interested in going a step further, we recommend (28) for a general introduction to cryptography, and (20; 21) for a rigorous and in-depth study of the foundations of cryptography.

2 Secure Multiparty Computation – Background and Definitions

2.1 Motivation and Highlights

Distributed computing considers the scenario where a number of distinct, yet connected, computing devices (or parties) wish to carry out a joint computation of some function. For example, these devices may be servers who hold a distributed database system, and the function to be computed may be a database update of some kind. The aim of *secure multiparty computation* is to enable parties to carry out such distributed computing tasks in a secure manner. Whereas distributed computing classically deals with questions of computing under the threat

of machine crashes and other inadvertent faults, secure multiparty computation is concerned with the possibility of deliberately malicious behavior by some adversarial entity. That is, it is assumed that a protocol execution may come under "attack" by an external entity, or even by a subset of the participating parties. The aim of this attack may be to learn private information or cause the result of the computation to be incorrect. Thus, two important requirements on any secure computation protocol are *privacy* and *correctness*. The privacy requirement states that nothing should be learned beyond what is absolutely necessary; more exactly, parties should learn their output and nothing else. The correctness requirement states that each party should receive its correct output. Therefore, the adversary must not be able to cause the result of the computation to deviate from the function that the parties had set out to compute.

The setting of secure multiparty computation encompasses tasks as simple as coin-tossing and broadcast, and as complex as electronic voting, electronic auctions, electronic cash schemes, contract signing, anonymous transactions, and private information retrieval schemes. Consider for a moment the tasks of voting and auctions. The privacy requirement for an election protocol ensures that no parties learn anything about the individual votes of other parties; the correctness requirement ensures that no coalition of parties has the ability to influence the outcome of the election beyond simply voting for their preferred candidate. Likewise, in an auction protocol, the privacy requirement ensures that only the winning bid is revealed (if this is desired); the correctness requirement ensures that the highest bidder is indeed the winning party (and so the auctioneer, or any other party, cannot bias the outcome). Due to its generality, the setting of secure multiparty computation can model almost every cryptographic problem.

To be even more concrete, let us consider the two-party problem of securely computing the median. Here, we have two parties with separate input sets X and Y. The aim of the parties is to jointly compute the median of the union of their sets $X \cup Y$, without revealing anything about each other's set that cannot be derived from the output itself.[1] Here, the parties' private inputs are X and Y, respectively, and their output is the median of $X \cup Y$. In order to obtain this output, they run an interactive protocol which involves them sending messages to each other according to some prescribed specification, which in turn should result in them learning the output as desired.

Security in multiparty computation. As we have mentioned above, the model that we consider is one where an adversarial entity controls some subset of the parties and wishes to attack the protocol execution. The parties under the control of the adversary are called corrupted, and follow the adversary's instructions. Secure protocols should withstand any adversarial attack (the exact power of the adversary will be discussed later). In order to formally claim and prove that a protocol is secure, a precise definition of security for multiparty computation is required. A number of different definitions have been proposed and these definitions aim to ensure a number of important security properties that are general enough to capture most (if not all) multiparty computation tasks. We now describe the most central of these properties:

- *Privacy:* No party should learn anything more than its prescribed output. In particular,

[1] Note that some information may be learned from the output. For example, if the median of the union is a number that is smaller than all of the values in X, and the sets are of the same size, then the parties will know that all of values in Y are smaller than all of the values in X. Nevertheless, this is "allowed" because it is inherent in the problem description – the parties must learn the output.

the only information that should be learned about other parties' inputs is what can be derived from the output itself. For example, in an auction where the only bid revealed is that of the highest bidder, it is clearly possible to derive that all other bids were lower than the winning bid. However, this should be the only information revealed about the losing bids.

- *Correctness:* Each party is guaranteed that the output that it receives is correct. To continue with the example of an auction, this implies that the party with the highest bid is guaranteed to win, and no party including the auctioneer can alter this.
- *Independence of Inputs:* Corrupted parties must choose their inputs independently of the honest parties' inputs. This property is crucial in a sealed auction, where bids are kept secret and parties must fix their bids independently of others. We note that independence of inputs is *not* implied by privacy. For example, it may be possible to generate a higher bid without knowing the value of the original one. Such an attack can actually be carried out on some encryption schemes (i.e., given an encryption of \$100, it is possible to generate a valid encryption of \$101, without knowing the original encrypted value).
- *Guaranteed Output Delivery:* Corrupted parties should not be able to prevent honest parties from receiving their output. In other words, the adversary should not be able to disrupt the computation by carrying out a "denial of service" attack.
- *Fairness:* Corrupted parties should receive their outputs if and only if the honest parties also receive their outputs. The scenario where a corrupted party obtains output and an honest party does not should not be allowed to occur. This property can be crucial, for example, in the case of contract signing. Specifically, it would be very problematic if the corrupted party received the signed contract and the honest party did not.

We stress that the above list does *not* constitute a definition of security, but rather a set of requirements that should hold for any secure protocol. Indeed, one possible approach to defining security is to generate a list of separate requirements (as above) and then say that a protocol is secure if all of these requirements are fulfilled. However, this approach is not satisfactory for the following reasons. First, it may be possible that an important requirement was missed. This is especially true because different applications have different requirements, and we would like a definition that is general enough to capture all applications. Second, the definition should be simple enough so that it is trivial to see that *all* possible adversarial attacks are prevented by the proposed definition.

The standard definition today (cf. (8) following (22; 4; 35)) therefore formalizes security in the following general way. As a mental experiment, consider an "ideal world" in which an external trusted (and incorruptible) party is willing to help the parties carry out their computation. In such a world, the parties can simply send their inputs to the trusted party, who then computes the desired function and passes to each party its prescribed output. Since the only action carried out by a party is that of sending its input to the trusted party, the only freedom given to the adversary is in choosing the corrupted parties' inputs. Notice that all of the above-described security properties (and more) hold in this ideal computation. For example, privacy holds because the only message ever received by a party is its output (and so it cannot learn any more than this). Likewise, correctness holds since the trusted party cannot be corrupted; thus it will always compute the function correctly.

Of course, in the "real world" there is no external party that can be trusted by all parties. Rather, the parties run some protocol among themselves without any help. Despite this, a

secure protocol should emulate the so-called "ideal world". That is, a real protocol that is run by the parties (in a world where no trusted party exists) is said to be secure, if no adversary can do more harm in a real execution than in an execution that takes place in the ideal world. This can be formulated by saying that for any adversary carrying out a successful attack in the real world, there exists an adversary that successfully carries out the same attack in the ideal world. However, successful adversarial attacks *cannot* be carried out in the ideal world. We therefore conclude that all adversarial attacks on protocol executions in the real world must also fail.

More formally, the security of a protocol is established by comparing the outcome of a real protocol execution to the outcome of an ideal computation. That is, for any adversary attacking a real protocol execution, there exists an adversary attacking an ideal execution (with a trusted party) such that the input/output distributions of the adversary and the participating parties in the real and ideal executions are essentially the same. Thus, a real protocol execution "emulates" the ideal world. This formulation of security is called the ideal/real simulation paradigm. In order to motivate the usefulness of this definition, we describe why all the properties described above are implied. Privacy follows from the fact that the adversary's output is the same in the real and ideal executions. Since the adversary learns nothing beyond the corrupted party's outputs in an ideal execution, the same must be true for a real execution. Correctness follows from the fact that the honest parties' outputs are the same in the real and ideal executions, and from the fact that in an ideal execution, the honest parties all receive correct outputs as computed by the trusted party. Regarding independence of inputs, notice that in an ideal execution, all inputs are sent to the trusted party before any output is received. Therefore, the corrupted parties know nothing of the honest parties' inputs at the time that they send their inputs. In other words, the corrupted parties' inputs are chosen independently of the honest parties' inputs, as required. Finally, guaranteed output delivery and fairness hold in the ideal world because the trusted party always returns all outputs. The fact that it also holds in the real world again follows from the fact that the honest parties' outputs are the same in the real and ideal executions.

We remark that the above informal definition is actually "overly ideal" and needs to be relaxed in settings where the adversary controls a half or more of the participating parties (i.e., in the case that there is no honest majority). When this number of parties is corrupted, it is known that it is *impossible* to obtain general protocols for secure multiparty computation that guarantee output delivery and fairness (e.g. (11)). Therefore, the definition is relaxed and the adversary is allowed to abort the computation (i.e., cause it to halt before termination), meaning that "guaranteed output delivery" is not fulfilled. Furthermore, the adversary can abort after it has already obtained its output, but before all the honest parties have received their outputs. Thus, "fairness" is not achieved. Loosely speaking, the relaxed definition is obtained by modifying the ideal execution and giving the adversary the additional capability of instructing the trusted party to not send outputs to some of the honest parties. Otherwise, the definition remains identical, thus preserving all other properties.

Adversarial power. The above informal definition of security omits one very important issue: the power of the adversary that attacks a protocol execution. As we have mentioned, the adversary controls a subset of the participating parties in the protocol. However, we have not described the corruption strategy (i.e., when or how parties come under the "control" of the adversary), the allowed adversarial behavior (i.e., does the adversary passively gather information or can it instruct the corrupted parties to act maliciously), and what complexity

the adversary is assumed to be (i.e., is it polynomial-time or computationally unbounded). We now describe the main types of adversaries that have been considered:

1. **Corruption strategy:** The corruption strategy deals with the question of when and how parties are corrupted. There are two main models:

   (a) Static corruption model: In this model, the adversary is given a fixed set of parties whom it controls. Honest parties remain honest throughout, while corrupted parties remain corrupted.

   (b) Adaptive corruption model: Rather than having a fixed set of corrupted parties, adaptive adversaries are given the capability of corrupting parties during the computation. The choice of who to corrupt and when can be arbitrarily decided by the adversary and may depend on its view of the execution ;for this reason, it is called adaptive. This strategy models the threat of an external "hacker" breaking into a machine during an execution. We note that in this model, once a party is corrupted, it remains corrupted from that point on.

   An additional model, called the proactive model (40; 9), considers the possibility that parties are corrupted only for a certain period of time. Thus, honest parties may become corrupted throughout the computation (like in the adaptive adversarial model), but may later become honest again.

2. **Allowed adversarial behavior:** Another parameter that must be defined relates to the actions that corrupted parties are allowed to take. Once again, there are two main types of adversaries:

   (a) Semi-honest adversaries: In the semi-honest adversarial model, even corrupted parties correctly follow the protocol specification. However, the adversary obtains the internal state of all the corrupted parties (including the transcript of all the messages received), and attempts to use this to learn information that should remain private. This is a rather weak adversarial model. However, there are some settings where it can realistically model threats to the system. Semi-honest adversaries are also called "honest-but-curious" and "passive".

   (b) Malicious adversaries: In this adversarial model, the corrupted parties can *arbitrarily* deviate from the protocol specification, according to the adversary's instructions. In general, providing security in the presence of malicious adversaries is preferred, as it ensures that no adversarial attack can succeed. Malicious adversaries are also called "active".

3. **Complexity:** Finally, we consider the assumed computational complexity of the adversary. As above, there are two categories here:

   (a) Polynomial-time: The adversary is allowed to run in (probabilistic) polynomial-time (and sometimes, expected polynomial-time). The specific computational model used differs, depending on whether the adversary is uniform (in which case, it is a probabilistic polynomial-time Turing machine) or non-uniform (in which case, it is modeled by a polynomial-size family of circuits). We remark that probabilistic polynomial-time is the standard notion of "feasibly" computation; any attack that cannot be carried out in polynomial-time is not a threat in real life.

   (b) Computationally unbounded: In this model, the adversary has no computational limits whatsoever.

The above distinction regarding the complexity of the adversary yields two very different models for secure computation: the *information-theoretic* model (7; 10) and the *computational* model (45; 18). In the information-theoretic setting, the adversary is not bound to any complexity class (and in particular, is not assumed to run in polynomial-time). Therefore, results in this model hold unconditionally and do not rely on any complexity or cryptographic assumptions. The only assumption used is that parties are connected via ideally *private* channels (i.e., it is assumed that the adversary cannot eavesdrop or interfere with the communication between honest parties).

By contrast, in the computational setting the adversary is assumed to run in polynomial-time. Results in this model typically assume cryptographic assumptions, such as the existence of trapdoor permutations. These are assumptions on the hardness of solving some problem (e.g., factoring large integers) whose hardness has not actually been proven but is widely conjectured.[2] We note that it is not necessary here to assume that the parties have access to ideally private channels, because such channels can be implemented using public-key encryption. However, it is assumed that the communication channels between parties are authenticated; that is, if two honest parties communicate, then the adversary can eavesdrop but cannot modify any message that is sent. Such authentication can be achieved using digital signatures (24) and a public-key infrastructure.

We remark that all possible combinations of the above types of adversaries have been considered in the literature.

Feasibility of secure multiparty computation. The above-described definition of security seems to be very restrictive in that no adversarial success is tolerated. Thus, one may wonder whether it is even possible to obtain secure protocols under this definition, and if yes, for which distributed computing tasks. Perhaps surprisingly, powerful feasibility results have been established, demonstrating that in fact, *any* distributed computing task can be securely computed. We now briefly state the most central of these results; let m denote the number of participating parties and let t denote a bound on the number of parties that may be corrupted:

1. For $t < m/3$ (i.e., when less than a third of the parties can be corrupted), secure multiparty protocols with fairness and guaranteed output delivery can be achieved for any function in a point-to-point network and without any setup assumptions. This can be achieved both in the computational setting (18) (assuming the existence of enhance trapdoor permutations[3]), and in the information-theoretic (private channel) setting (7; 10).
2. For $t < m/2$ (i.e., in the case of a guaranteed honest majority), secure multiparty protocols with fairness and guaranteed output delivery can be achieved for any function assuming that the parties have access to a broadcast channel. This can be achieved in the computational setting (18) (with the same assumptions as above), and in the information-theoretic setting (43).
3. For $t \geq m/2$ (i.e., when the number of corrupted parties is not limited), secure multiparty protocols (without fairness or guaranteed output delivery) can be achieved assuming that the parties have access to a broadcast channel and that enhanced trapdoor permutations

[2] The unfortunate state of affairs is that the ability to unconditionally prove the hardness of solving such problems would involve major breakthroughs in the area of computational complexity, and as such, seems currently out of reach.

[3] see (21, Appendix C)

exist (45; 18; 21). These feasibility results hold only in the computational setting; analogous results for the information-theoretic setting cannot be obtained when $t \geq m/2$ (7).

In summary, secure multiparty protocols exist for any distributed computing task. In the computational model, this holds for all possible numbers of corrupted parties, with the qualification that when no honest majority exists, fairness and guaranteed output delivery are not obtained. We note that the above results all hold with respect to *malicious adversaries*. (The status regarding adaptive versus static adversaries is more involved and is therefore omitted here.)

Challenges for secure multiparty computation. Given the aforementioned results, it may seem that there are no challenges remaining for the field of secure multiparty computation. This is far from the truth. In particular, the above are all *feasibility results*, meaning that their focus is on establishing that secure computation (with some set of parameters) can be achieved. However, if secure protocols are to be used in practice, protocols must be designed that are highly efficient. This is especially true when the protocols are to be used on very large data sets. In such cases, requiring a relatively heavy number-theoretic computation[4] per bit of the input is completely infeasible (although computation on this scale is often considered to be efficient in cryptography). We remark that there are many other open issues in the area of secure multiparty computation (e.g., secure protocol composition in general network settings). However, we will focus on issues of efficiency in this paper.

2.2 Definitions of Security

We will present two definitions below: one for the case of semi-honest adversaries and one for the case of malicious adversaries. In both cases, we focus on the setting of *static corruptions* and *no honest majority*. We will also only consider polynomial-time adversaries. For the sake of clarity we present definitions for the two-party case only; the generalization to multiparty computation is straightforward.

Technical Preliminaries

We denote the security parameter by n; essentially, this parameter determines the length of cryptographic keys (or more exactly, the length of input needed to some hard problem so that real-world adversaries cannot break the problem in a reasonable amount of time). We say that a function $\mu(\cdot)$ is negligible in n (or just negligible) if for every positive polynomial $p(\cdot)$ there exists an integer N such that for all $n > N$ it holds that $\mu(n) < 1/p(n)$. Note that an event that happens with negligible probability happens so infrequently that we can effectively dismiss it.

All parties, including the adversary, run in time that is polynomial in n. We assume that each party has a "security parameter tape" that is initialized to the string of n ones, denoted 1^n; the parties then run in time that is polynomial in the input written on that tape. (The reason why the security parameter is received as 1^n rather than just the binary representation of n is due to a technicality—we want the party to run in time that is polynomial in the *length* of its input. Thus, we make the length of the security parameter input n by setting it to be 1^n; this technicality can be ignored.)

[4] A typical such computation is that of a modular exponentiation, computing $x^a \bmod N$, where x, a and N are all very large numbers, on the scale of hundreds of digits each.

Let $X(n,a)$ and $Y(n,a)$ be random variables indexed by n and a (n here denotes the security parameter and a will typically represent the inputs to the protocol), and let $X = \{X(n,a)\}_{n\in\mathbb{N},a\in\{0,1\}^*}$ and $Y = \{Y(n,a)\}_{n\in\mathbb{N},a\in\{0,1\}^*}$ be distribution ensembles. We say that these two random variables are computationally indistinguishable if no algorithm running in polynomial-time can tell them apart (except with negligible probability). More precisely, we say that X and Y are computationally indistinguishable, denoted $X \stackrel{c}{\equiv} Y$, if for every non-uniform polynomial-time distinguisher D there exists a function $\mu(\cdot)$ that is negligible in n, such that for *every* $a \in \{0,1\}^*$,

$$|\Pr[D(X(n,a)) = 1] - \Pr[D(Y(n,a)) = 1]| < \mu(n)$$

Thus, if X and Y are indistinguishable, it holds that for every efficient distinguisher D and for every positive polynomial $p()$, there exists an N such that for all $n > N$ it holds that D cannot distinguish between the two with probability better than $1/p(n)$. Therefore, X and Y are the same for all intents and purposes. Typically, the distributions X and Y will denote the output vectors of the parties in real and ideal executions, respectively. In this case, a denotes the parties' inputs. (The outputs of the parties are modeled as random variables since the operation of the parties is typically probabilistic, depending on random coin tosses (or random inputs) used by the parties.)

Security in the Presence of Semi-Honest Adversaries

The model that we consider here is that of two-party computation in the presence of *static semi-honest* adversaries. Such an adversary controls one of the parties (statically, and so at the onset of the computation) and follows the protocol specification exactly. However, it may try to learn more information than allowed by looking at the transcript of messages that it received. The definitions presented here are according to Goldreich in (21).

Two-party computation. A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs $x, y \in \{0,1\}^n$, the output-pair is a random variable $(f_1(x,y), f_2(x,y))$ ranging over pairs of strings. The first party (with input x) wishes to obtain $f_1(x,y)$, and the second party (with input y) wishes to obtain $f_2(x,y)$. We often denote such a functionality by $(x,y) \mapsto (f_1(x,y), f_2(x,y))$. For example, consider the oblivious transfer functionality where the first party has a pair of strings (x_0, x_1) for input and the second party has a bit $\sigma \in \{0,1\}$. The aim of the protocol is for the second party to receive the message x_σ (but it should learn nothing about $x_{1-\sigma}$ and the first party should learn nothing about σ). This functionality is specified by $((x_0,x_1),\sigma) \mapsto (\lambda, x_\sigma)$, where λ denotes the empty string (specifying in this case that the sender learns nothing). When the functionality f is probabilistic, we sometimes use the notation $f(x,y,r)$, where r is a uniformly chosen random tape used for computing f.

Privacy by simulation. Intuitively, a protocol is secure if whatever can be computed by a party participating in the protocol can be computed based on its input and output only. This is formalized according to the simulation paradigm. Loosely speaking, we require that a party's *view* in a protocol execution be simulatable given only its input and output. We remark that in the semi-honest model, this definitional approach is equivalent to the ideal/real model approach described above; see (21). This then implies that the parties learn nothing from the protocol *execution* itself, as desired.

Definition of security. We begin with the following notation:

- Let $f = (f_1, f_2)$ be a probabilistic polynomial-time functionality and let π be a two-party protocol for computing f.
- The view of the i^{th} party ($i \in \{1,2\}$) during an execution of π on input (x,y) and security parameter n is denoted $\mathsf{view}_i^\pi(n,x,y)$ and equals $(1^n, x, r^i, m_1^i, ..., m_t^i)$, where r^i equals the contents of the i^{th} party's *internal* random tape, and m_j^i represents the j^{th} message that it received.
- The output of the i^{th} party during an execution of π on input (x,y) and security parameter n is denoted $\mathsf{output}_i^\pi(n,x,y)$ and can be computed from its own view of the execution. Denote

$$\mathsf{output}^\pi(n,x,y) = (\mathsf{output}_1^\pi(n,x,y), \mathsf{output}_2^\pi(n,x,y)).$$

Note that $\mathsf{view}_i^\pi(n,x,y)$ and $\mathsf{output}^\pi(n,x,y)$ are random variables, with the probability taken over the random tapes of all the parties.

In the definition below we quantify only over inputs x and y that are of the same length. Some restriction on input lengths is required, and padding can be used to achieve this restriction; see discussion in (21).

Definition 1. *(security w.r.t. semi-honest behavior):* Let $f = (f_1, f_2)$ be a functionality. We say that π securely computes f in the presence of static semi-honest adversaries if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that for every $x, y \in \{0,1\}^*$ where $|x| = |y|$, we have:

$$\{(S_1(1^n, x, f_1(x,y)), f(x,y))\}_{n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \{(\mathsf{view}_1^\pi(n,x,y), \mathsf{output}^\pi(n,x,y))\}_{n\in\mathbb{N}} \quad (1)$$

$$\{(S_2(1^n, y, f_2(x,y)), f(x,y))\}_{n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \{(\mathsf{view}_2^\pi(n,x,y), \mathsf{output}^\pi(n,x,y))\}_{n\in\mathbb{N}} \quad (2)$$

Equations (1) and (2) state that the view of a party can be simulated by a probabilistic polynomial-time algorithm given access to the party's *input and output only*. This can be seen by the fact that S_1 is given x and $f(x,y)$ and must generate output that is indistinguishable from the view of P_1 in a real execution. We note that it is not enough for the simulator S_i to generate a string indistinguishable from $\mathrm{view}_i^\pi(n,x,y)$. Rather, the *joint distribution* of the simulator's output and the functionality output $f(x,y)$ must be indistinguishable from $(\mathsf{view}_i^\pi(n,x,y), \mathsf{output}^\pi(n,x,y))$. This is necessary for probabilistic functionalities; see (8; 21) for a full discussion.

A simpler formulation for deterministic functionalities. In the case that the functionality f is deterministic, a simpler definition can be used. We refer to (21, Section 7.2.2) for more discussion.

Security in the Presence of Malicious Adversaries

In this section, we consider *malicious adversaries* who may arbitrarily deviate from the specified protocol. When considering malicious adversaries and the case of no honest majority (as in the important case of two parties), there are certain undesirable actions that cannot be prevented.

Specifically, a party may refuse to participate in the protocol, may substitute its local input (and instead use a different input), or may abort the protocol prematurely. One ramification of the adversary's ability to abort is that it is impossible to achieve "fairness". That is, the adversary may obtain its output while the honest party does not. These adversarial capabilities are therefore not prevented by the definition of security (formally, they are "allowed" by incorporating them in the ideal execution as well). The definition below is formalized according to the ideal/real model paradigm described above.

Execution in the ideal model. Let P_1 and P_2 be the parties and let I denote the indices of the corrupted parties controlled by an adversary $\mathcal{A}$. In principle, it is possible for zero, one, or both parties to be corrupted. However, for the sake of simplicity, we will consider the most important case, that either $I = \{1\}$ or $I = \{2\}$ (i.e., exactly one of the two parties is corrupted). An ideal execution proceeds as follows:

Inputs: Each party obtains an input; the i^{th} party's input is denoted x_i. The adversary $\mathcal{A}$ receives an auxiliary input denoted z.

Send inputs to trusted party: The honest party P_j for $j \notin I$ sends its input x_j to the trusted party. The corrupted party P_i for $i \in I$ (who is controlled by $\mathcal{A}$) may either abort by replacing the input x_i with a special abort message, send its input x_i, or send some other input of the same length to the trusted party. This decision is made by $\mathcal{A}$ and may depend on the value x_i for $i \in I$ and its auxiliary input z. Denote the inputs sent to the trusted party by (w_1, w_2) (note that w_i does not necessarily equal x_i).

If the trusted party receives an input of the form abort from P_i, it sends abort to both parties and the ideal execution terminates. Otherwise, the execution proceeds to the next step.

Trusted party sends outputs to adversary: The trusted party computes the pair of outputs $(f_1(w_1, w_2), f_2(w_1, w_2))$ and sends $f_i(w_1, w_2)$ to the corrupted party P_i.

Adversary instructs trusted party to continue or halt: $\mathcal{A}$ sends either continue or abort to the trusted party. If it sends continue, the trusted party sends $f_j(w_1, w_2)$ to the honest party P_j. Otherwise, if $\mathcal{A}$ sends abort, the trusted party sends abort to party P_j.

Outputs: The honest party always outputs the message it obtained from the trusted party. The corrupted party outputs nothing. The adversary $\mathcal{A}$ outputs any arbitrary (probabilistic polynomial-time computable) function of the initial input x_i, the auxiliary input z, and the output abort or $f_i(w_1, w_2)$ obtained from the trusted party.

Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a two-party functionality, where $f = (f_1, f_2)$; let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $I = \{1\}$ or $I = \{2\}$ be the index of the corrupted party. Then, the ideal execution of f on inputs (x_1, x_2), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{IDEAL}_{f,\mathcal{A}(z),I}(n, x_1, x_2)$), is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

Execution in the real model. We next consider the real model in which a real two-party protocol π is executed (and there exists no trusted third party). In this case, the adversary $\mathcal{A}$ sends all messages in place of the corrupted party, and may follow an arbitrary polynomial-time strategy. In contrast, the honest party follows the instructions of π. (We assume that at least one of the parties is honest, since we are not required to help a party that deviates from the protocol; therefore, if both parties are corrupt we are not required to provide any security guarantee.)

Let f be as above and let π be a two-party protocol for computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the index of the corrupted party. Then, the real execution of π on inputs (x_1, x_2), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{REAL}_{\pi,\mathcal{A}(z),I}(n, x_1, x_2)$, is defined as the output vector of the honest party and the adversary $\mathcal{A}$ from the real execution of π. The auxiliary input z models side information that the adversary may have and that is important for obtaining meaningful notions of security (in reality, the adversary may know part of the input or may at least know what inputs the honest party is unlikely to have; such knowledge is auxiliary information and is included in z).

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate executions of the real-model protocol. As in the semi-honest case, we will consider executions where the inputs are of the same length.

Definition 2. *(security w.r.t. malicious adversaries):* Let f and π be as above. Protocol π is said to securely compute f with abort in the presence of malicious adversaries if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic expected polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every I, every $x_1, x_2 \in \{0,1\}^*$ such that $|x| = |y|$, and every auxiliary input $z \in \{0,1\}^*$:

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),I}(n, x_1, x_2)\right\}_{n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(n, x_1, x_2)\right\}_{n\in\mathbb{N}}$$

We remark that the ideal-model adversary is denoted $\mathcal{S}$ because in security proofs it behaves as a *simulator* (simulating a real protocol execution for $\mathcal{A}$ while it really interacts in the ideal model). We also allow the ideal-model adversary to run in *expected* polynomial-time (rather than strict polynomial-time), because this is often necessary for proving the security of efficient protocols.

Notice that the security guarantees provided by Definition 2 are very strong. Essentially, the adversary's only possible attacks are to choose its input as it wishes (arguably, a legitimate strategy), and cause an early abort in the protocol. In this light, the feasibility results that we surveyed in Section 2.1 are truly amazing.

Modular Sequential Composition

Sequential composition theorems for secure computation are important for two reasons. First, they constitute a security goal within themselves. Second, they are useful tools that help in writing proofs of security. The basic idea behind these composition theorems is that it is possible to design a protocol that uses an ideal functionality as a subroutine, then analyze the security of the protocol when a trusted party computes this functionality. For example, assume that a protocol is constructed that uses the secure computation of some functionality as a subroutine. First, we construct a protocol for the functionality in question and prove its security. Next, we prove the security of the larger protocol that uses the functionality as a subroutine in a model where the parties have access to a trusted party computing the functionality. The composition theorem then states that when the "ideal calls" to the trusted party for the functionality are replaced by real executions of a secure protocol computing this functionality, the protocol remains secure.

The f-hybrid model. The aforementioned composition theorems are formalized by considering a *hybrid model*, where both parties interact with each other (as in the real model) and use trusted help (as in the ideal model). Specifically, the parties run a protocol π that contains ideal calls to a trusted party computing a functionality f. These ideal calls are just instructions to send an input to the trusted party. Upon receiving the output back from the trusted party, the protocol π continues. We stress that honest parties do not send messages in π between the time that they send input to the trusted party and the time that they receive back output (this is because here, *sequential* composition is considered). Of course, the trusted party may be used a number of times throughout the π-execution. However, each time is independent (i.e., the trusted party does not maintain any state between these calls). We call the regular messages of π that are sent amongst the parties standard messages and the messages that are sent between parties and the trusted party ideal messages.

Let f be a functionality and let π be a two-party protocol that uses ideal calls to a trusted party computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the f-hybrid execution of π on inputs (x_1, x_2), auxiliary input z to $\mathcal{A}$, and security parameter n, denoted $\text{HYBRID}^{f}_{\pi,\mathcal{A}(z),I}(n, x_1, x_2)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of π, with a trusted party computing f.

Sequential modular composition. Let f and π be as above, and let ρ be a protocol. Consider the real protocol π^{ρ} that is defined as follows. All standard messages of π are unchanged. When a party P_i is instructed to send an ideal message α_i to the trusted party, it begins a real execution of ρ with input α_i instead. When this execution of ρ concludes with output β_i, party P_i continues with π as if β_i was the output received by the trusted party (i.e., as if it were running in the f-hybrid model). The following theorem was proven in (8):

Theorem 3. Let f be a two-party probabilistic polynomial-time functionality and let ρ be a protocol that securely computes f with abort in the presence of malicious (resp., semi-honest) adversaries. Let g be a two-party functionality, and let π be a protocol that securely computes g with abort in the f-hybrid model in the presence of malicious (resp., semi-honest) adversaries. Then, π^{ρ} securely computes g with abort in the presence of malicious (resp., semi-honest) adversaries.

The use of this composition theorem (and others similar to it) greatly simplifies proofs of security. Instead of analyzing a large protocol and proving reductions to subprotocols, it suffices to analyze the security of the large protocol in the idealized hybrid model.

Other Definitions of Security

The semi-honest and malicious models described above are the most standard. However, to some extent, both of these models are problematic. First of all, the semi-honest model is too weak for many settings. Would we accept an election protocol where any party who actively cheats can sway the outcome, or learn individual parties' votes? Thus, this model is more appropriate for settings where the participating parties basically trust each other. However, they may have other reasons for not joining all of their data. This is the case of hospitals who wish to carry out joint research on their confidential patient records. Due to privacy laws they are *not allowed* to look at each others' databases; however, they are basically honest. Thus, the semi-honest model is very suitable for such settings. We remark that the security guaranteed in this model ensures that no inadvertent leakage of information takes place. In particular, even if one of the hospital's computers is broken into after the protocol execution

takes place, it is guaranteed that nothing is revealed about the other databases. Having said this, in many (if not most) cases of privacy-preserving data mining, we do not and cannot trust the participants to not cheat (especially if they can easily do so without getting caught). This leads to the conclusion that the malicious model is preferable, and indeed from a security point of view this is definitely the case. However, when it comes to *efficiency*, the malicious model poses great difficulties. In particular, although we already know that in principle every efficient function can be securely computed in this model, until recently we have known of very few interesting functions (e.g., functions that constitute non-trivial data-mining algorithms) that can be securely computed in the presence of malicious adversaries and that are even reasonably efficient. Thus, it is not clear if our goal of constructing privacy-preserving data mining protocols that can be used in practice will be achieved if we limit ourselves to this model. Below, we briefly describe two different definitions of security that provide lower guarantees, but are arguably sufficient in many settings. Importantly, in both of these models it is possible to construct efficient secure protocols.

Guaranteeing privacy only. As we have discussed, the definition of security that follows the ideal/real simulation paradigm provides strong security guarantees. In particular, it guarantees privacy, correctness, independence of inputs, and so on. However, in some settings, it may be sufficient to guarantee privacy only. We warn that this is not so simple, and in many cases it is difficult to separate privacy from correctness. Nevertheless, privacy still provides a non-trivial security guarantee. We will not present a general definition here, because this depends heavily on the function being computed. Nevertheless, we will present a definition for one specific function in order to demonstrate how such definitions look. For this purpose, we consider the oblivious transfer function. In this function, there is a sender S with a pair of input strings (x_0, x_1) and a receiver R with an input bit σ. The output of the function is nothing for the sender and the string x_σ for the receiver. Thus, a secure oblivious transfer protocol has the property that the sender learns nothing about σ while the receiver learns at most one of the strings (x_0, x_1). Unfortunately, defining privacy here without resorting to the ideal model is very non-trivial. Specifically, it is easy to define privacy in the presence of a malicious sender S^*; we simply say that S^* cannot distinguish the case that R has input 0 from the case that it has input 1. However, it is more difficult to define privacy in the presence of a malicious receiver R^*, because it does learn something. A naive approach to defining this says that for some bit b it holds that R^* knows nothing about x_b. However, this value of b may depend on the messages sent during the oblivious transfer and so cannot be fixed ahead of time. Fortunately, in the case of two-message oblivious transfer (where the receiver sends one message and the sender replies with one message), it is possible to formally define this. The following definition of security for oblivious transfer is based on (25) and states that replacing one of x_0 and x_1 with some other x should go unnoticed by the receiver. The question of which of x_0, x_1 to replace causes a problem which is solved in the case of a two-message protocol by fixing the first message; see below. (In the definition below we use the following notation: for a two-party protocol with parties S and R, we denote by $view^n_S(S(a), R(b))$ the view of S in an execution where it has input a, and R has input b, and the security parameter is n. Furthermore, we denote by $S_n(a; q)$ the distribution over the message sent by S upon input a, security parameter n, and message received q. When the protocol has two messages only and the first message q is sent by R, the message $S(a; q)$ defines R's view in the execution.)

Definition 4. A **two-message** two-party probabilistic polynomial-time protocol (S, R) is said to be a private oblivious transfer if the following holds:

- *Correctness:* If S and R follow the protocol, then after an execution in which S has for

input any pair of strings $x_0, x_1 \in \{0,1\}^n$ and R has for input any bit $\sigma \in \{0,1\}$, the output of R is x_σ.

- *Privacy for R:* For every non-uniform probabilistic polynomial-time S^* and every auxiliary input $z \in \{0,1\}^*$, it holds that

$$\{view^n_{S^*}(S^*(z), R(0))\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \{view^n_{S^*}(S^*(z), R(1))\}_{n\in\mathbb{N}} \ .$$

- *Privacy for S:* For every non-uniform deterministic polynomial-time receiver R^*, every auxiliary input $z \in \{0,1\}^*$, and every triple of inputs $x_0, x_1, x \in \{0,1\}^n$ it holds that either:

$$\{S_n((x_0, x_1); R^*(z))\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \{S_n((x_0, x); R^*(z))\}_{n\in\mathbb{N}}$$

or

$$\{S_n((x_0, x_1); R^*(z))\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \{S_n((x, x_1); R^*(z))\}_{n\in\mathbb{N}} \ .$$

The way to view the above definition of privacy for S is that R^*'s first message, denoted $R^*(z)$ determines whether it should receive x_0 or x_1. If it determines, for example, that it should receive x_0, then the distribution over S's reply when its input is (x_0, x_1) is indistinguishable from the distribution when its input is (x_0, x). Clearly this implies that R^* cannot learn anything about x_1 when it receives x_0 and vice versa.

Note that when defining the privacy for S we chose to focus on a *deterministic* polynomial-time receiver R^*. This is necessary in order to fully define the message $R^*(z)$ for any given z, which in turn fully defines the string x_b, that $R^*(z)$ does *not* learn. By making R^* non-uniform, we see that this does not weaken the adversary (since R^*'s advice tape can hold its "best coins"). We remark that generalizing this definition to protocols that have more than two messages is non-trivial.

The above example demonstrates that it is possible to define "privacy only" for secure computation. However, it also demonstrates that this task can be *very difficult*. In general, when a party does not receive output, this task is easy. However, when a party does receive output, defining privacy without resorting to the ideal model is problematic (and often unclear as to how it can be achieved).

We conclude with one important remark regarding "privacy-only" definitions. As we have mentioned, an important property of security definitions is a composition theorem that guarantees certain behavior when the secure protocol is used as a subprotocol in another larger protocol. No such general composition theorems are known for definitions that follow the privacy-only approach. As such, this approach has a significant disadvantage.

Security in the presence of covert adversaries (3). Recently, a new adversarial model was introduced that lies between the semi-honest and malicious models. The motivation behind the definition is that in many real-world settings, adversaries are willing to actively cheat (and as such are not semi-honest), but only if they are not caught (and as such they are not arbitrarily malicious). This is the case in many business, financial, political and diplomatic settings, where honest behavior cannot be assumed, but where the companies, institutions, and individuals involved cannot afford the embarrassment, loss of reputation, and negative press associated with being *caught* cheating. Clearly, with such adversaries, it may be the case that the risk of being caught is weighed against the benefits of cheating, and it cannot be assumed that players would avoid being caught at any price and under all circumstances. Accordingly, the definition explicitly models the probability of catching adversarial behavior. The definition

of security is based on the classical ideal/real simulation paradigm with the following difference: for a value $0 < \epsilon \leq 1$ (called the *deterrence factor*), the definition guarantees that any attempt to "cheat" by an adversary is detected by the honest parties with probability at least ϵ. Thus, provided that ϵ is sufficiently large, an adversary that wishes not to be caught cheating will refrain from *attempting* to cheat, lest it be caught doing so. Note that the security guarantee does not preclude successful cheating. Indeed, if the adversary decides to cheat then it may gain access to the other parties' private information or bias the result of the computation. The only guarantee is that if it attempts to cheat, then there is a fair chance that it will be caught doing so. The above motivation is formulated within the ideal/real paradigm and also has the benefits of a sequential composition theorem. Importantly, it has also been shown that efficient protocols can be constructed under this definition; see (3) for more details.

3 Secure Multiparty Computation – Constructions

In this section, we survey some of the known constructions and techniques for building secure protocols. With one exception, we do not provide proofs of security but rather present very basic intuition as to why security is achieved. We warn that such intuitive arguments are in no way to be accepted as justification for the security of a protocol. We only allow ourselves this privilege here because *all* of the protocols that we present have been rigorously proven secure in the papers that present them.

3.1 Basic Building Blocks

We describe here some simple protocols that are often used as basic building blocks, or primitives, of secure computation protocols. The protocols we describe here include oblivious transfer and oblivious polynomial evaluation, which are two-party protocols, and homomorphic encryption, which is an encryption system with special properties.

Oblivious Transfer

Oblivious transfer is a simple functionality involving two parties. It is a basic building block of many cryptographic protocols for secure computation. (In fact, it was shown by Kilian (29) that by using an implementation of oblivious transfer, and no other cryptographic primitive, it is possible to construct any secure computation protocol.)

We will use a specific variant of oblivious transfer, 1-out-of-2 oblivious transfer, which was suggested by Even, Goldreich and Lempel (14) (as a variant of a different but equivalent type of oblivious transfer that has been suggested by Rabin (42)). The protocol involves two parties, a *sender* and a *receiver*; its functionality is defined as follows:

- **Input:** *The sender's input is a pair of strings* (x_0, x_1) *and the receiver's input is a bit* $\sigma \in \{0, 1\}$.
- **Output:** *The receiver's output is* x_σ *(and nothing else), while the sender has no output.*

In other words, 1-out-of-2 oblivious transfer implements the function $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$, where λ denotes the empty string (i.e., no output).

Oblivious transfer protocols have been designed based on virtually all known assumptions

which are used to construct specific trapdoor functions (i.e., public key cryptosystems), and also based on generic assumptions such as the existence of enhanced trapdoor permutations. There are simple and efficient protocols for oblivious transfer which are secure only against semi-honest adversaries (14; 21). In particular, one straightforward approach (14) is for the receiver to generate two random public keys, a key P_σ whose decryption key it knows, and a key $P_{1-\sigma}$ whose decryption key it does not know.[5] The receiver then sends these two keys to the sender, which encrypts x_0 with the key P_0 and encrypts x_1 with the key P_1, and sends the two results to the receiver. The receiver can then decrypt x_σ but not $x_{1-\sigma}$. Intuitively, it is obvious that the sender does not learn anything about σ, since its view in the protocol can be easily simulated: the only message it receives includes two random public keys. As for the sender's privacy, if the receiver follows the protocol, it only knows one private key and can therefore only decrypt one of the inputs. We also assume the encryption scheme to be semantically secure (23).[6] Therefore, in the simulation, given the receiver's input σ and its output x_σ, we can send it a message containing an encryption of σ with the public key P_σ and an encryption of a random value using the public key $p_{1-\sigma}$. The receiver cannot distinguish the second value from an encryption of $x_{1-\sigma}$, since it does not know the corresponding private key.

It is a little more challenging to construct oblivious transfer protocols which are secure against malicious adversaries. In order to adapt the oblivious transfer protocol described above we must ensure that the receiver chooses the public keys appropriately. This can be done using zero-knowledge proofs that are used by the receiver to prove that it chooses the keys correctly. Fortunately, there are very efficient proofs for this case: an efficient two-message oblivious transfer protocol following this paradigm was presented by Bellare and Micali and proved secure in the random oracle model (6). Oblivious transfer protocols with similar overhead that provide *privacy* against malicious adversaries (as in Definition 4), were presented in (2; 36) (based on the Decisional Diffie-Hellman assumption), and later generalized in (25) to use smooth projective hashing with a special property (and as a result can be based on the Nth Residuosity assumption, or the Quadratic Residuosity assumption).

Oblivious transfer is often the most computationally intensive operation of secure protocols, and is repeated many times. Each invocation of oblivious transfer typically requires a constant number of invocations of trapdoor permutations (i.e., public-key operations, or exponentiations). It is possible to reduce the amortized overhead of oblivious transfer to one exponentiation per a logarithmic number of oblivious transfers, even in the case of malicious adversaries (36). Further, one can extend oblivious transfer in the sense that one has to compute, in advance, a small number of oblivious transfers. This then allows one to compute an essentially unlimited number of of transfers at the cost of computing hash functions alone (26; 39). (All these results are proved secure in the random oracle model. In this model, it is assumed that the parties have access to an external party who computes a random function for them. Once a protocol is proven secure under this assumption, the external random function is replaced by some concrete cryptographic hash function, and the security of the concrete scheme follows from heuristic arguments about the random-looking behavior of the hash function. See (28, Section 13.1) for a detailed discussion about this model.)

[5] The actual secure protocol is different because we don't always know how to sample a public key without knowing its secret key. Nevertheless, this gives the flavor of the construction.

[6] Namely, even if it known that an encryption is of one of only two possible messages m_0, m_1, it is infeasible to identify the plaintext with probability significantly better than $1/2$. See (23) for a precise definition.

We note that some of the oblivious transfer protocols described above are proved secure using definitions which are weaker than Definition 2 (namely they are not proven according to the real/ideal simulation paradigm but can be proven under a definition like Definition 4). When such protocols are used as primitives in other protocols, it is not possible to simply plug them into Theorem 3 and analyze security in the hybrid model. Rather, more intricate security proofs are required in order to prove security.

An efficient oblivious transfer protocol. We now present the protocol of (36) and prove that it achieves *privacy*, as formalized in Definition 4. (Readers who are not interested in the details of the implementation of oblivious transfer can proceed to Section 27.)

Protocol 5.

- **Input:** The sender has a pair of strings (m_0, m_1) and the receiver has a bit σ.
- **Auxiliary input:** The parties have the description of a group $\mathcal{G}$ of order n, and a generator g for the group; the order of the group is known to both parties.
- **The protocol:**

  1. The receiver R chooses $a, b, c \in_R \{0, \ldots, n-1\}$ and computes γ as follows:

     (a) If $\sigma = 0$ then $\gamma = (g^a, g^b, g^{ab}, g^c)$

     (b) If $\sigma = 1$ then $\gamma = (g^a, g^b, g^c, g^{ab})$

     R sends γ to S.

  2. Denote the tuple γ received by S by (x, y, z_0, z_1). Then, S checks that $z_0 \neq z_1$. If they are equal, it aborts outputting $\perp$. Otherwise, S chooses random $u_0, u_1, v_0, v_1 \in_R \{0, \ldots, n-1\}$ and computes the following four values:

     $$w_0 = x^{u_0} \cdot g^{v_0} \qquad k_0 = (z_0)^{u_0} \cdot y^{v_0}$$
     $$w_1 = x^{u_1} \cdot g^{v_1} \qquad k_1 = (z_1)^{u_1} \cdot y^{v_1}$$

     S then encrypts m_0 under k_0 and m_1 under k_1. For the sake of simplicity, assume that one-time pad type encryption is used. That is, assume that m_0 and m_1 are mapped to elements of $\mathcal{G}$. Then, S computes $c_0 = m_0 \cdot k_0$ and $c_1 = m_1 \cdot k_1$ where multiplication is in the group $\mathcal{G}$.

     S sends R the pairs (w_0, c_0) and (w_1, c_1).

  3. R computes $k_\sigma = (w_\sigma)^b$ and outputs $m_\sigma = c_\sigma \cdot (k_\sigma)^{-1}$.

The security of Protocol 5 rests on the decisional Diffie-Hellman (DDH) assumption that states that tuples of the form (g^a, g^b, g^c) where $a, b, c \in_R \{0, \ldots, n-1\}$ are indistinguishable from tuples of the form (g^a, g^b, g^{ab}) where $a, b \in_R \{0, \ldots, n-1\}$ (recall that n is the order of the group $\mathcal{G}$ that we are working in). This implies that an adversarial S^* cannot discern whether the message sent by R is (g^a, g^b, g^{ab}, g^c) or (g^a, g^b, g^c, g^{ab}) and so R's input is hidden from S^*. The motivation for S's privacy is more difficult and it follows from the fact that – informally speaking – the exponentiations computed by S completely randomize the triple (g^a, g^b, g^c). Interestingly, it is still possible for R to derive the key k_σ that results from the randomization of (g^a, g^b, g^{ab}). None of these facts are evident from the protocol itself but are demonstrated below in the proof. We therefore proceed directly to prove the following theorem:

Theorem 6. Assume that the decisional Diffie-Hellman problem is hard in $\mathcal{G}$ with generator g. Then, Protocol 5 is a private oblivious transfer, as in Definition 4.

Proof. The first requirement of Definition 4 is correctness, and we prove this first. Let m_0, m_1 be S's input and let σ be R's input. The message sent by S to R is $w_\sigma = x^{u_\sigma} \cdot g^{v_\sigma}$ and $c_\sigma = m_\sigma \cdot k_\sigma$ where $k_\sigma = (z_\sigma)^{u_\sigma} \cdot y^{v_\sigma}$. Correctness follows from the fact that:

$$(w_\sigma)^b = x^{u_\sigma \cdot b} \cdot g^{v_\sigma \cdot b} = g^{(a \cdot b) \cdot u_\sigma} \cdot g^{b \cdot v_\sigma} = z_\sigma^{u_\sigma} \cdot y^{v_\sigma} = k_\sigma$$

where the third equality is due to the fact that $z_\sigma = g^{ab}$. Thus, R recovers the key k_σ and can compute $(k_\sigma)^{-1}$ and $m_\sigma = c_\sigma \cdot (k_\sigma)^{-1}$.

Next, we prove the requirement of privacy for R. Recall that this requirement is that S^*'s view when R has input 0 is indistinguishable from its view when R has input 1. Now, the view of an adversarial sender S^* in Protocol 5 consists merely of R's first message γ. By the DDH assumption, we have that

$$\left\{(g^a, g^b, g^{ab})\right\}_{a,b \in_R \{1,\ldots,n-1\}} \stackrel{c}{\equiv} \left\{(g^a, g^b, g^c)\right\}_{a,b,c \in_R \{1,\ldots,n-1\}}$$

Now, assume by contradiction that there exists a probabilistic polynomial-time distinguisher D and a non-negligible function ϵ such that

$$\left|\Pr[D(g^a, g^b, g^{ab}, g^c) = 1] - \Pr[D(g^a, g^b, g^c, g^{ab}) = 1]\right| \geq \epsilon(n)$$

where $a, b, c \in_R \{0, \ldots, n-1\}$. Then, by subtracting and adding $\Pr[D(g^a, g^b, g^c, g^d) = 1]$ we have:

$$\begin{aligned}
&\left|\Pr[D(g^a, g^b, g^{ab}, g^c) = 1] - \Pr[D(g^a, g^b, g^c, g^{ab}) = 1]\right| \\
&\quad \leq \left|\Pr[D(g^a, g^b, g^{ab}, g^c) = 1] - \Pr[D(g^a, g^b, g^c, g^d) = 1]\right| \\
&\qquad + \left|\Pr[D(g^a, g^b, g^c, g^d) = 1] - \Pr[D(g^a, g^b, g^c, g^{ab}) = 1]\right|
\end{aligned}$$

where $a, b, c, d \in_R \{0, \ldots, n-1\}$.

Therefore,

$$\left|\Pr[D(g^a, g^b, g^{ab}, g^c) = 1] - \Pr[D(g^a, g^b, g^c, g^d) = 1]\right| \geq \frac{\epsilon(n)}{2} \tag{3}$$

or

$$\left|\Pr[D(g^a, g^b, g^c, g^d) = 1] - \Pr[D(g^a, g^b, g^c, g^{ab}) = 1]\right| \geq \frac{\epsilon(n)}{2} \ . \tag{4}$$

Assume that (3) holds. We construct a distinguisher D' for the DDH problem that works as follows. Upon input $\gamma = (x, y, z)$, the distinguisher D' chooses a random $d \in_R \{0, \ldots, n-1\}$ and hands D the tuple $\gamma' = (x, y, z, g^d)$. The key observation is that on the one hand, if $\gamma = (g^a, g^b, g^c)$, then $\gamma' = (g^a, g^b, g^c, g^d)$. On the other hand, if $\gamma = (g^a, g^b, g^{ab})$, then $\gamma' = (g^a, g^b, g^{ab}, g^d)$. Noting that in this last tuple c does not appear, with c and d distributed identically, we have $\gamma' = (g^a, g^b, g^{ab}, g^c)$. Thus,

$$\begin{aligned}
&\left|\Pr[D'(g^a, g^b, g^{ab}) = 1] - \Pr[D'(g^a, g^b, g^c) = 1]\right| \\
&\qquad = \left|\Pr[D(g^a, g^b, g^{ab}, g^c) = 1] - \Pr[D(g^a, g^b, g^c, g^d) = 1]\right| \\
&\qquad \geq \frac{\epsilon(n)}{2}
\end{aligned}$$

in contradiction to the DDH assumption. A similar analysis follows in the case that (4) holds. It therefore follows that ϵ must be a negligible function. The proof of R's privacy is concluded by noting that (g^a, g^b, g^{ab}, g^c) is exactly the distribution over R's message when $\sigma = 0$, and (g^a, g^b, g^c, g^{ab}) is exactly the distribution over R's message when $\sigma = 1$. Thus, the privacy of R follows from the DDH assumption over the group in question.[7]

However, it remains to prove privacy for the sender S. Let $\gamma = (x, y, z_0, z_1)$ denote $R^*(z)$'s first message, and let a and b be such that $x = g^a$ and $y = g^b$. If $z_0 = z_1$, then S sends nothing and so clearly the requirement in Definition 4 holds. Otherwise, let $\tau \in \{0, 1\}$ such that $z_\tau \neq g^{ab}$ (note that since $z_0 \neq z_1$ it cannot be that both $z_0 = g^{ab}$ and $z_1 = g^{ab}$). The sender S's security is based on the following claim:

Claim 3.1. Let $x = g^a$, $y = g^b$ and $z_\tau = g^c \neq g^{ab}$. Then, given a, b and c, the pair of values (w_τ, k_τ) where $w_\tau = x^{u_\tau} \cdot g^{v_\tau}$ and $k_\tau = (z_\tau)^{u_\tau} \cdot y^{v_\tau}$ is uniformly distributed when u_τ, v_τ are chosen uniformly in $\{0, \ldots, n-1\}$.

The proof of this claim is implicit in (38; 44) in the context of a "randomized reduction" of the DDH problem, and we will not reprove it here. Now, given this claim it follows that k_τ is uniformly distributed, even given w_τ (and $k_{1-\tau}, w_{1-\tau}$). Thus, $k_\tau \cdot m_\tau$ is distributed identically to $k_\tau \cdot m$ for every $m, m_\tau \in \mathcal{G}$. This completes the proof of the sender's privacy. ■

Homomorphic Encryption

A homomorphic encryption scheme is an encryption scheme which allows certain algebraic operations to be carried out on the encrypted plaintext, by applying an efficient operation to the corresponding ciphertext. In addition, we require in this paper that the encryption scheme be semantically secure (see Footnote 6). In particular, we will be interested in additively homomorphic encryption schemes where the message space is a ring (or, more commonly, a field). Here, there exists an efficient algorithm $+_{\text{pk}}$ whose input is the public key of the encryption scheme and two ciphertexts, and whose output is $E_{\text{pk}}(m_1) +_{\text{pk}} E_{\text{pk}}(m_2) = E_{\text{pk}}(m_1 + m_2)$. (Namely, it is easy to compute, given the public key alone and the encryption of the sum of the plaintexts of two ciphertexts.) There is also an efficient algorithm $\cdot_{\text{pk}}$, whose input consists of the public key of the encryption scheme, a ciphertext, and a constant c in the ring, and whose output is $c \cdot_{\text{pk}} E_{\text{pk}}(m) = E_{\text{pk}}(c \cdot m)$.

An efficient implementation of an additive homomorphic encryption scheme with semantic security was given by Paillier (41). In this cryptosystem, the encryption of a plaintext from $[1, N]$, where N is an RSA modulus, requires two exponentiations modulo N^2. Decryption requires a single exponentiation. The Damgård-Jurik cryptosystem (13) is a generalization of the Paillier cryptosystem which encrypts messages from the range $[1, N^s]$ using computations modulo N^{s+1}, where N is an RSA modulus and s a natural number. It enables more efficient encryption of larger plaintexts than Paillier's cryptosystem (which corresponds to the case $s = 1$). The security of both schemes is based on the decisional composite residuosity assumption.

[7] One may wonder why we bothered to provide such a detailed proof. After all, isn't it clear that if (g^a, g^b, g^{ab}) is indistinguishable from (g^a, g^b, g^c) then it must hold that (g^a, g^b, g^{ab}, g^c) is indistinguishable from (g^a, g^b, g^c, g^{ab})? Our answer to this is an unequivocal *NO*! Seemingly intuitive arguments in cryptography are often incorrect, thus, full proofs are necessary to avoid error.

Oblivious Polynomial Evaluation

The problem of oblivious polynomial evaluation (OPE) involves two parties, a sender and a receiver. The input of the sender is a polynomial Q of degree k over some finite field $\mathcal{F}$, namely a polynomial $Q(z) = \sum_{i=0}^{k} a_i z^i$ (the degree of the polynomial, which we will denote as k, is public). The input of the receiver is an element $z \in \mathcal{F}$. After the protocol is run, the receiver outputs $Q(z)$ without learning anything else about Q, while the sender learns nothing. In other words, OPE implements the functionality $(Q, z) \mapsto (\lambda, Q(z))$, where λ is the empty output. The major motivation for oblivious polynomial evaluation is the fact that the output of a k degree random polynomial is $(k+1)$-wise independent; this is very useful in the construction of cryptographic protocols. Another motivation is that polynomials can be used for approximating functions that are defined over the Real numbers.

The OPE problem was introduced in (37), where an efficient solution based on oblivious transfer was also presented. We will briefly describe a simpler protocol based on homomorphic encryption (this protocol is secure in the semi-honest model and achieves privacy—but not simulatable security—in the face of a malicious adversary). This protocol works in the following way: The receiver defines a homomorphic encryption system with semantic security for which only the receiver knows the decryption key. The receiver then sends the encryptions $E(z), E(z^2), \ldots, E(z^k)$ to the sender. The sender uses the homomorphic properties to compute $E(Q(z)) = (a_k \cdot_{\mathrm{pk}} E(z^k)) +_{\mathrm{pk}} \cdots +_{\mathrm{pk}} (a_1 \cdot_{\mathrm{pk}} E(z)) +_{\mathrm{pk}} E(a_0)$, and sends this encrypted value to the receiver. The receiver decrypts it and obtains $Q(z)$. This protocol requires $O(k)$ communication and computation. It is secure against semi-honest adversaries since it is easy to simulate the views of each of the parties given their inputs and outputs (in particular, the sender only sees encryptions that were carried out using a semantically secure encryption scheme).

3.2 Generic Constructions

There are generic protocols that implement secure computation for any probabilistic polynomial-time function. These protocols are different for a scenario in which there are two parties, and for the multiparty scenario where there are $m > 2$ parties.

The Two-Party Case

Secure computation in the two-party case can be efficiently implemented by a generic protocol due to Yao (45). The protocol (or rather, simple variants of it) are proved to be secure, according to Definitions 1 and 2, against both semi-honest and malicious adversaries (32; 33).

Denote the two parties participating in the protocol as Alice (A) and Bob (B), and denote their respective inputs by x and y. Let f be the function that they wish to compute (for simplicity, assume that Bob alone learns the value $f(x, y)$). The protocol is based on expressing f as a combinatorial circuit with gates expressing any function $g : \{0,1\} \times \{0,1\} \to \{0,1\}$ (including simple `or`, `and`, and `not` gates). Note that it is known that any polynomial-time function can be expressed as a combinatorial circuit of polynomial size. The input to the circuit consists of the bits of the inputs x and y, and the output of the circuit is the value $f(x, y)$.

The protocol is based on evaluating this circuit. The number of rounds of the protocol is constant. Its communication overhead depends on the size of the circuit, while its computation overhead depends on the number of input wires (more specifically, it requires running one oblivious transfer protocol for every input wire of party B, and, in addition computing efficient

symmetric encryption/decryption functions for each gate of the circuit). A more detailed analysis of the overhead of the protocol is given below. More details on the protocol, including a proof of security, can be found in (32). We now provide a high level description of Yao's protocol.

Encoding the circuit. Yao's protocol works by having one of the parties (say, Alice) first generate an "encrypted" or "garbled" circuit computing f, and send its representation to Bob. The encrypted circuit is generated in the following way:

- First, Alice "hardwires" her input into the circuit, generating a circuit computing $f(x, \cdot)$.
- Alice then assigns to each wire i of the circuit two random ("garbled") values (W_i^0, W_i^1), corresponding to values 0 and 1 of the wire (the random values should be long enough to be used as keys to a symmetric encryption scheme, e.g., 128 bits long).
- For every gate g (say, computing the value of wire k as a function of wires i and j) Alice prepares a table T_g that encrypts the garbled value of the output wire using the garbled values of the two input wires as keys. The table has four entries, one entry for every combination of input values, and each entry contains the encryption of the garbled value of the corresponding value of the output wire. For example, in an `or` gate, the garbled input values (W_i^0, W_j^1) are used as keys for encrypting the garbled output value W_k^1. (The encryption is carried out using a semantically secure symmetric encryption scheme. Note that such schemes can be realized in practice by primitives such as block ciphers and are therefore very efficient.) The table enables computation of the *garbled* output of g, from the garbled inputs to g. Furthermore, given the two garbled inputs to g, the table does not disclose information about the output of g for any other inputs, nor does it reveal the values of the actual input bits.

Before proceeding, we present a concrete example of a garbled AND gate. Let the input wires to the gate be labeled 1 and 2, and let the output wire be labeled 3. We then choose 6 random 128-bit strings (symmetric encryption keys) $W_1^0, W_1^1, W_2^0, W_2^1, W_3^0, W_3^1$ and compute the following four encryptions, based on the gate's truth table:

Wire 1 input	**Wire 2 input**	**Wire 3 output**	**Garbled value**
0	0	0	$E_{W_1^0}(E_{W_2^0}(W_3^0))$
0	1	0	$E_{W_1^0}(E_{W_2^1}(W_3^0))$
1	0	0	$E_{W_1^1}(E_{W_2^0}(W_3^0))$
1	1	1	$E_{W_1^1}(E_{W_2^1}(W_3^1))$

Observe that when the output value is 0 the key that is encrypted is W_3^0, and when the output value is 1 the key that is encrypted is W_3^1. Furthermore, given W_1^0 and W_1^1 (for example), it is possible to correctly decrypt $E_{W_1^0}(E_{W_2^1}(W_3^0))$ to obtain W_3^0, but it is impossible to decrypt any other entry of the table. We remark that with the encryption scheme used, it is possible for the decryption process to detect that it has received an incorrect decryption. Thus, given the one garbled value per input wire and the four encryptions of the garbled gate in random order, Bob attempts to decrypt them all. Exactly one will decrypt correctly, and he will thus receive exactly one of the garbled values on the output wire (but not the other). The interesting property here is that Bob receives the correct garbled value for the output wire without knowing anything about the computation he has carried out. In particular, he has

no idea what values are associated with the garbled values and what gate he has computed. Nevertheless, it is guaranteed that he has correctly computed the gate. As we have mentioned above, garbled gates are prepared for all gates in the circuit.

The representation of the circuit includes the wiring of the original circuit (namely, a mapping from inputs or gate outputs to gate inputs), the tables T_g, and the tables that translate the garbled values of the *output* wires of the circuit to actual 0/1 values. In this form, the representation reveals nothing but the wiring of the circuit, and therefore Bob learns nothing from this stage. (We assume that the wiring of the circuit is not secret, which is obviously the case if the function f is public and the only secret information of Alice is her input x. If the wiring of the circuit implementing f is secret and is known only to Alice, it can be encoded by representing the circuit as part of Alice's input and letting the parties evaluate a universal circuit, i.e., a circuit whose input is $(\langle f, x\rangle, y)$ and whose output is $f(x, y)$.)

Encoding Bob's input. The tables described above enable the computation of the garbled output of any gate from its garbled inputs. Therefore, given these tables and the garbled values of the input wires of the circuit, it is possible to compute the garbled values of the output wires of the circuit and then translate them to actual values. In order for Bob to compute the circuit, he must obtain the garbled values of all input wires. This is achieved by having Bob and Alice run a 1-out-of-2 oblivious transfer protocol for each input wire of Bob. In these protocol executions, Alice is the sender, and her inputs are the two garbled values of this wire; Bob is the receiver, and his input is his input bit associated with that wire. As a result of the oblivious transfer protocol, Bob learns the garbled value of his input bit and learns nothing about the garbled value of the other bit, and Alice learns nothing.

Computing the circuit. The full protocol for computing the circuit starts by letting the parties execute the oblivious transfer stage. Afterwards, Alice sends Bob the description of the garbled circuit as detailed above, and the garbled values of her input wires. Bob now has sufficient information to compute the output of the circuit on his own. After computing $f(x, y)$, he can send this value to Alice if she requires it.

To show that the protocol is secure we must prove that the parties learn nothing that cannot be computed in the ideal model, namely computed based on the input and output only. This proof is provided in (32) for the case of semi-honest adversaries (and in (33) for a variant of this protocol, which handles the case of malicious adversaries). The main observation regarding the security of each gate is that the function used to encrypt gate entries ensures that without knowledge of the correct keys, i.e., garbled values of input wires, the encrypted values of the gate look random. Therefore, for every gate it holds that knowledge of one garbled value of each of the input wires discloses only a single one of the four key combinations of garbled input values, and therefore only a single garbled output value of the gate; Bob cannot distinguish the other garbled value from random. As for the security of the complete circuit, the oblivious transfer protocol ensures that Bob learns only a single garbled value for each input wire, and Alice does not learn which value it was. Inductively, Bob can compute only a single garbled output value of each gate, and in particular, only a single output of the circuit. We stress that the method in which the tables were constructed hides the values of intermediate results (i.e., of gates inside the circuit).

It is possible to adapt the protocol for circuits in which gates have more than two inputs, and even for wires with more than two possible values (which is possible since there is no need

for a physical realization of the circuit; this might enable the construction of more compact circuits). The size of the table for a gate with ℓ inputs, where each input can have d values is d^ℓ.

Overhead. The overhead of the protocol described above involves: (1) Alice and Bob engaging in an oblivious transfer protocol for every input wire of the circuit that is associated with Bob's input, (2) Alice sending Bob tables of size that are linear in the size of the circuit, and (3) Bob decrypting a constant number of ciphertexts for every gate of the circuit (this is the cost incurred in evaluating the gates).

The computation overhead is dominated by the oblivious transfer stage, since the evaluation of the gates uses symmetric encryption which is very efficient compared to oblivious transfers that require modular exponentiations (this holds for small circuits; if the circuit is large then the circuit computation may begin to dominate). The computation overhead is therefore roughly linear in the length of Bob's input. The number of rounds of the protocol is constant (namely, the variant described here has two rounds using the two-round oblivious transfer protocols of (14; 21; 36)). The communication overhead is linear in the size of the circuit. (The variant of the protocol described in (33), which provides security against malicious adversaries, requires sending s copies of the circuit in order to limit the probability of cheating to be exponentially small in s. See also (27) for a different variant, which provides security against malicious adversaries at the cost of applying public key operations for every gate.)

A major factor dominating the overhead is, therefore, the size of the circuit representation of f. There are many functions for which we do not know how to create linear size circuits (e.g., functions computing multiplications or exponentiations, or functions that use indirect addressing). However, there are many other functions, notably those involving additions and comparisons, which can be computed by linear size circuits. The size of the input should also be reasonable. For example, we cannot expect that two parties, each of them holding a database with millions of entries, could run the protocol for computing a function whose inputs are the entire databases.

We note that an implementation of Yao's protocol exists (the Fairplay project (34)). This system receives as input a description of a function in a high-level language, and generates a circuit computing it, along with two programs, one for each of the parties, implementing Yao's protocol for this circuit.

The Multiparty Case

As we described in Section 2.1, there are well known constructions which enable a set of $m > 2$ parties to compute any function of their inputs without revealing any other information. As with Yao's protocol, these constructions are based on expressing the function as a circuit and applying a secure computation protocol to this circuit. These protocols are pretty efficient if the resulting circuit is of reasonable size, but they do have some drawbacks compared to the two-party protocol. For example, some of the protocols require public-key operations (rather than symmetric key operations) for every gate of the circuit, some have a number of rounds which is linear in the size of the circuit (rather than a constant number of rounds), all protocols require communication between every pair of the m participating parties, and some of them require the use of a broadcast channel.

The multiparty construction of Goldreich, Micali, and Wigderson (18) is based on describing

the function as a binary circuit (or rather, a circuit with addition and multiplication gates over $GF[2]$), and starting from a state in which each party knows a share of each input wire. The protocol requires every pair of parties to run a short computation (e.g., an oblivious transfer) for each multiplication gate of the circuit. The number of rounds is therefore linear in the depth of the circuit, and the communication is $O(m^2k|C|)$ for a circuit C and security parameter k. If the number of corrupt parties t is smaller than $m/3$, then the construction provides security against malicious adversaries, with fairness and guaranteed delivery. If $t < m/2$, this level of security can be achieved if there is access to a broadcast channel. Otherwise ($m/2 \leq t < m$), security can be provided against malicious adversaries, but without fairness and guaranteed delivery. There exists a construction, due to Beaver, Micali, and Rogaway (5), which runs in a constant number of rounds. Like the construction of (18), it is based on assuming the existence of trapdoor permutations. However, the construction is somewhat more intricate: it includes a first stage in which the parties jointly construct garbled tables for each gate, and a second stage in which these gates are evaluated without additional communication.

The constructions of Ben-Or, Goldwasser, and Wigderson (7), and of Chaum, Crépeau, and Damgård (10) are based on the assumption that a private channel exists between every pair of parties (in this respect they are different than the constructions described above, which are based on cryptographic assumptions). We will describe here the basic properties of the construction of (7) by first describing the function as an arithmetic circuit over an arbitrary ring, with addition and multiplication gates (note that Binary circuits are a special case of arithmetic circuits). The protocol starts with each party knowing a share (over the ring) of each input wire, and ends with each party knowing a share of each output wire. Addition gates are computed locally by every party, while multiplication gates require each pair of parties to exchange a message. This results, like in the protocol of (18), in a number of rounds which is linear in the depth of the circuit. It also results in the total communication of $O(m^2k|C|)$. However, unlike the protocols of (18; 5), there is no need to compute public-key operations. Instead, the computation involves only simple additions and multiplications. Security against malicious adversaries is provided as long as $t < m/3$, and against semi-honest adversaries as long as $t < m/2$.

Threshold decryption. Threshold decryption is an example of a multiparty functionality. The setting includes m parties and an encryption scheme. It is required that any $m' < m$ of the parties is able to decrypt messages, while any coalition of strictly less than m' parties learns nothing about encrypted messages. This functionality can, of course, be implemented using generic constructions, but there are specific constructions implementing it for almost any encryption scheme, and these are far more efficient than applying the generic constructions to compute this functionality. Interestingly, threshold decryption of homomorphic encryption can be used as a primitive for constructing a very efficient generic protocol for secure multiparty computation, with a communication overhead of only $O(mk|C|)$ bits (see (16) for a construction secure against semi-honest adversaries, and (12) for a construction secure against malicious adversaries).

3.3 Specialized Constructions

Although generic constructions for secure computation can, in principle, efficiently compute any polynomial function, the resulting overhead is often unacceptable. This might be due to the size of the circuit computing the function, or to the fact that each input value (or sometimes,

as in the two-party case, each input bit) incurs expensive operations such as input sharing or computing an oblivious transfer. In general, when considering semi-honest adversaries and a reasonably sized circuit, the protocols are reasonably efficient. However, when considering malicious adversaries these protocols are typically not practical even for small circuits.

We describe in this section three specialized constructions which are considerably more efficient than applying generic constructions to the same functions. The constructions we describe are secure against semi-honest adversaries, although for some of them there exist variants which are secure against malicious adversaries. The constructions are based on the use of homomorphic encryption, oblivious polynomial evaluation, and the reduction of the computed function to simpler functionalities, with the analysis of the resulting protocol in the hybrid model (as in Theorem 3). For each function, we describe the overhead of applying a generic construction, then describe the basic details of the specialized construction and its overhead.

Set Intersection

Consider two parties, Alice and Bob, who each have a set of k items drawn from a large domain of size N. Denote Alice's and Bob's sets as $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$, respectively. The parties wish to compute the intersection of their two sets, without revealing any other information. This problem was investigated in (17) and is denoted as the set intersection problem or as the private matching problem (the multiparty case was investigated in (30)).

Private equality test. As a warm-up, consider a simpler case where each party has a single item ($k = 1$). The function outputs 1 if the inputs of the two parties are equal (namely, $x = y$), and 0 otherwise. A simple way to implement this function is to use Yao's protocol, applied to a circuit which compares each bit of Alice's input to the corresponding bit of Bob's input and outputs the result of applying an `and` operator to the results of these comparisons. The circuit is of reasonable size, $O(\log N)$ gates, and therefore the resulting protocol is quite efficient. There are several alternative solutions to this problem, with similar overhead (15; 37).

Let us also describe another solution to the private equality test problem, which is based on the use of homomorphic encryption: Let Alice define a homomorphic encryption system for which only she knows the private key. She then encrypts x and sends the encryption $E(x)$ to Bob. Bob chooses a random value r and uses the homomorphic properties to compute $(E(x) +_{\mathrm{pk}} E(-y)) \cdot_{\mathrm{pk}} r = E((x-y) \cdot r)$, and sends this result back to Alice. Alice decrypts this message, and outputs 1 if and only if the decrypted value is equal to 0. Note that if $x = y$, then Bob indeed sends Alice an encryption of 0. If, on the other hand, $x \neq y$, Bob sends to Alice an encryption of a random value (generated by multiplying $x - y$ by the random value r). Alice, therefore, does not learn anything about y except for whether it is equal to x. (This is proved according to Definition 1, by showing that given Alice's input and output, it is easy to simulate her view in the protocol. Indeed, if the output is $x = y$, we know that the message that Alice receives is an encryption of 0, and if the output is $x \neq y$, the message she receives is an encryption of a random value.) Bob does not learn anything about x, since we assume the encryption system to be semantically secure.

Set intersection. Solving the set intersection problem is more involved than private equality testing since each item in Alice's set might be equal to any of Bob's items. Therefore, a

simple reduction to private equality testing requires $O(k^2)$ comparisons (comparing each of Alice's inputs to all of Bob's inputs), which we would like to avoid. A straightforward circuit comparing the sets is of size $O(k^2 \log N)$ and an alternative method of similar complexity but using OPE was presented in (37). (There are also simple constructions which use only $O(k)$ public key operations, but are only proved in the random oracle model.)

An efficient protocol for set intersection, due to (17), can be based on homomorphic encryption: let Alice define a homomorphic encryption system for which only she knows the private key. Alice then defines a polynomial P of degree k whose roots are her inputs, namely

$$P(y) = (x_1 - y)(x_2 - y) \cdots (x_k - y) = \sum_{i=0}^{k} \alpha_i y^i.$$

The coefficients of the polynomial are $\alpha_0, \ldots, \alpha_k$. Alice then encrypts each of the coefficients of the polynomial and sends these encrypted values to Bob. Note that for each y_i in Bob's list he can compute

$$(E(\alpha_k) \cdot_{\mathrm{pk}} y_i^k) +_{\mathrm{pk}} (E(\alpha_{k-1}) \cdot_{\mathrm{pk}} y_i^{k-1}) +_{\mathrm{pk}} \cdots +_{\mathrm{pk}} (E(\alpha_1) \cdot_{\mathrm{pk}} y_i) +_{\mathrm{pk}} E(\alpha_0) = E(P(y_i)).$$

Bob will actually pick a random value r_i for each y_i in his list and compute $E(r \cdot P(y_i) + y_i)$. If y_i is equal to an element in Alice's list, then this is an encryption of y_i, whereas otherwise it is an encryption of a random element. Bob sends the k resulting encryptions to Alice, who decrypts them. If any of the decrypted values is in her input set, she decides that this value is in the intersection. It is easy to see that, as in the private equality test, no information is revealed except for the identities of the items in the intersection (the proof here, too, is by showing a simulation satisfying Definition 1).

The major computational overhead of this protocol is the multiplication of the homomorphically encrypted values by constants, which is implemented using exponentiations, and is repeated $O(k^2)$ times in the protocol. The protocol can be changed, using hashing, to essentially require only $O(k)$ exponentiations and $O(k)$ communication (the exact asymptotic expression involves also a $\log \log k$ expression, but for any feasible value of k it is bounded by a small constant). Variants of the protocol exist which compute only the size of the intersection, or which indicate if the size of the intersection is greater than some threshold. The basic protocol is secure only against semi-honest adversaries. There is also a variant which is secure against malicious adversaries, but it is analyzed in the random oracle model (see (17) for details of all these variants).

Computing the Median

Assume now that Alice and Bob each have a list of n *distinct* numerical values from a domain of size N. They wish to compute the median of the union of their two lists while revealing no other information. More generally, they might wish to compute the kth ranked element (i.e, the kth largest element) in this union (the median is the case where $k = n$ because overall there are $2n$ values). Applying any of the generic constructions requires using all input bits of each of the parties, therefore yielding an overhead of at least $\Omega(n \log N)$. This overhead might be too large if the parameters are large (say, if the lists include millions of items). We will describe here a protocol, due to (1), which has a sublinear overhead of $O(\log n \log N)$ (or $O(\log k \log N)$ in the general case).

The protocol is based on reducing the computation of the median to $\log n$ secure comparisons of $\log N$ bit numbers. Namely, it is a reduction to a simpler protocol in which each party

has a private input, and the output is 1 if and only if Bob's item is greater than Alice's item. (This function is known as the millionaires problem. Applying Yao's generic protocol to solve it results in a protocol which uses $O(\log N)$ oblivious transfers and $O(\log N)$ communication.)

The protocol for computing the median works in the following way: Alice and Bob separately compute the median value of their own lists, which we will denote as m_A and m_B (we assume that the lengths of the list are powers of two, and define the median of a list of length $n = 2^i$ to be the item ranked 2^{i-1} in the list). The parties run the secure comparison protocol to find out if $m_A < m_B$. If this is the case, Alice removes all items smaller or equal to m_A from her list, while Bob removes all items in his list which are greater than m_B. If the result is that $m_A > m_B$, then each party removes the other half of his/her list. It is easy to see that the length of the lists is reduced by a factor of 2 using this computation. It is also straightforward to verify that the median of the union of the two original lists is guaranteed to be in the short lists, which remain after this step (this observation holds since every item that is removed is guaranteed to be smaller than more than half of the items, or greater than at least half of the items). Given these two observations, we apply this computation again to the new lists, repeating this step $\log 2n$ times until we are left with a single item, which is guaranteed to be the median of the union of the original lists.

The analysis above establishes the correctness and the overhead of the protocol. We should also convince ourselves of its security. It is sufficient to show that, assuming the number of elements held by each party is public information, Alice (and similarly Bob), given its own input and the value of the median, can simulate the execution of the protocol in the hybrid model, where the comparisons are carried out by a trusted party (the proof follows Theorem 3—the composition theorem). Consider the simulation of the first step of the protocol: we know Alice's input and final output—the median of the union of the lists. Since this value must be in the lists which are retained for the second step, we can easily deduce the result of the first comparison. Namely, if the median is strictly greater than m_A, then $m_A < m_B$ (therefore Alice removes all items smaller or equal to m_A). Otherwise $m_A > m_B$. We can therefore simulate the first step. Similarly, we can simulate all steps of the protocol. A similar argument holds for Bob's part in the protocol. The interested reader can consult (1) for a detailed description and analysis of the protocol, and also for variants for the case of malicious adversaries, and for the multiparty case.

Computing ID3

ID3 is a basic algorithm for constructing decision trees, which are a tool for solving the classification problem in machine learning and data mining. The input to a classification problem is a structured database in which each row represents a *transaction* and each column is an *attribute* taking on different values (for example, each row could represent a patient, and each column a different symptom). One of the attributes in the database is designated as the *class* attribute (e.g., it could denote whether the patient has a certain disease). The goal is to use the database in order to predict the class of a new transaction by viewing only the non-class attributes.

A decision tree is a rooted tree in which each internal node corresponds to an attribute, and the edges leaving it correspond to the possible values taken on by that attribute. The leaves of the tree contain the *expected* class value for transactions matching the path from the root to that leaf. Given a decision tree, one can predict the class of a new transaction by traversing the nodes from the root down. The value of the leaf at the end of this path is the expected

class value of the new transaction.

The ID3 algorithm is used to design a decision tree based on a given database. The tree is constructed top-down in a recursive fashion. At the root, each attribute is tested to determine how well it alone classifies the transactions. The "best" attribute is then chosen, and the remaining transactions are partitioned by it. ID3 is then recursively called on each partition, i.e., on a smaller database containing only the appropriate transactions, and without the splitting attribute.

The central principle of ID3 is to choose the *best* predicting attribute by checking which attribute reduces the information (in the information-theoretic sense) of the class-attribute to the greatest degree. Namely, to choose the attribute that maximizes the *information gain*, defined as the difference between the entropy of the class attribute and the entropy of the class attribute given the value of the chosen attribute. This decision rule results in a greedy algorithm that searches for a small decision tree consistent with the database. (Note that we only discuss the basic ID3 algorithm, and assume that each attribute is categorical and has a fixed set of possible values.)

Privacy preserving distributed computation of ID3. The setting we examine involves two parties, each with a database of different transactions, where all transactions have the same set of attributes (this scenario is also denoted as a "horizontally partitioned" database). The parties wish to compute a decision tree by applying the ID3 algorithm to the union of their databases. An efficient privacy preserving protocol for this problem was described in (31). We describe its basic details below, and refer the reader to (31) for the complete solution.

Applying Yao's generic protocol encounters some major obstacles: the size of the databases is typically very large (e.g., it is common to have millions of transactions) and invoking an oblivious transfer protocol per input bit is too costly. In addition, the circuit representation of ID3 is very large, since the basic step of the algorithm, repeated multiple times per node, involves computing the difference between two entropy values (each defined as $\sum p_i \log(p_i)$, where each p_i is the fraction of transactions in which the class attribute, and possibly other attributes, have certain values). Computing the logarithm function, which is defined over the Real numbers, is also problematic, since most cryptographic protocols compute functions over finite fields. Running ID3 also involves many rounds, with each round depending on the results of the previous rounds; therefore a naive circuit implementation could require an encoding of many copies of each step, each corresponding to a specific result of the previous rounds.

Computing ID3. The secure protocol is based on the observation that each node of the tree can be computed separately, with the output made public, before continuing to the next node. This is true since the assignments of attributes to each node are part of the output and may therefore be revealed. The computation starts from the root of the tree. Once the attribute of a given node has been found, both parties can separately partition their remaining transactions accordingly for the coming recursive calls. As a result, the protocol is reduced to privately finding the attribute of a node, namely the attribute with the highest information gain.

Let A be some attribute obtaining values $a_1, ..., a_m$ and let $T(a_j)$ be the subset of transactions obtaining value a_j for A. Let $T_A(a_j)$ and $T_B(a_j)$ be the corresponding subsets in Alice's and Bob's inputs (therefore, $T(a_j) = T_A(a_j) \cup T_B(a_j)$). The computation which quantifies the information gain in identifying the class of a transaction in T *given* the value of A involves expressions of the form $\frac{|T(a_j)|}{|T|}$ and $\log \frac{|T(a_j)|}{|T|}$, where $|T|$ is the size of the database. The value

$|T|$ is constant and can therefore be ignored, since we are only interested in comparing values to each other. The main challenge is computing the logarithm function, namely computing $\log(T_A(a_j) + T_B(a_j))$, where $T_A(a_j)$ is known to Alice and $T_B(b_j)$ is known to Bob. (More accurately, the parties compute two shares, Z_A and Z_B, which are random under the constraint that $Z_A + Z_B = \log(T_A(a_j) + T_B(a_j))$.) The logarithm function can be approximated using the Taylor approximation, which is essentially a polynomial. This computation can be securely computed using oblivious polynomial evaluation. The actual details of the protocol are quite intricate. We refer the interested reader to (31) for details.

4 Common Errors in Applications of Secure Computation

There are common errors which often occur when designing secure protocols. Protocols in which these errors exist cannot, of course, be proven secure according to the definitions of Section 2. There are, however, multiple examples of published protocols which suffer from these errors. We would like to use this section to highlight some of these errors in order to inform readers of common pitfalls in the design of secure protocols.

4.1 Semi-honest Behavior does not Preclude Collusions

Assuming that adversaries are semi-honest does not ensure that no two parties collude. The "semi-honest adversary" assumption merely ensures that an adversary follows the protocol and only tries to learn information from messages it received during protocol execution. It is still possible, however, that the adversary controls more than a single party and might use the information it learns from all the parties it controls.

Consider, for example, the following protocol run between n parties, denoted $P_1, \ldots, P_n$. The parties have private inputs $x_1, \ldots, x_n$ and they wish to compute the sum $x_1 + \cdots + x_n$. The protocol starts by P_1 choosing a random value r and sending $x_1 + r$ to P_2 (assume that the computations are done in a finite field). Each party P_i, for $2 \leq i < n$, receives a message m_i from P_{i-1}, and sends the message $m_{i+1} = m_i + x_i$ to P_{i+1}. Finally, P_n sends $m_n + x_n$ to P_1, who subtracts r from this value and publishes the result (which indeed equals $x_1 + \cdots + x_n$).

This protocol is indeed secure against semi-honest parties as long as no two parties collude. However, an adversary controlling parties P_i and P_j, where $j > i + 1$, can learn the sum $x_{i+1} + \cdots + x_{j-1}$ by computing $m_{j-1} - m_i$. This is something that cannot be learned in the ideal model (given only the overall sum), even when an adversary controls P_i and P_j, and thus this protocol is *not* secure.

4.2 Input Dependent Flow

Consider the following example: two parties run a protocol to decide if their two inputs are equal (assume that each input is of length k). The protocol works by running a simpler protocol which compares two bits. The input to this simpler protocol is a pair of bits taken from the same location in both inputs. The first comparison is of the most significant bits of both inputs, and afterwards successive bits are compared, until a difference is found or it is decided that the two inputs are equal. Note that this protocol executes only a single comparison if the two inputs differ in their most significant bit, but might perform more comparisons (say, k comparisons if the inputs differ only in their least significant bit). The protocol therefore

leaks information. If it exits after i comparisons the parties can conclude that the $i-1$ most significant bits of their inputs are equal. This is information that cannot be deduced in the ideal model where the parties are only told if the inputs are equal or not equal.

The source of the error in the protocol above is that the flow of the protocol (namely, the decision which parts of it to execute) depends on the private input of the parties. Consequently, the flow cannot be simulated by Alice, given her input and output alone (even if the output shows that Alice's input is different than Bob's, it does not tell her the first location in which the inputs differ). The protocol, therefore, is not secure according to Definition 1 (or any reasonable definition).

Note that the flow of execution in the protocol for computing the median and the protocol for computing ID3 (discussed above) depends on the results of previous computations in the protocol. (For example, in the median protocol the decision of which parts of the inputs will be removed depends on the result of the previous comparisons executed in the protocol.) However, in these protocols the control flow decisions can be simulated, given the output of the computed function (e.g., given the median value), and therefore they do not contradict the security of the protocol. It is important to note that sometimes it is not trivial to identify whether the flow of execution can be simulated by the output alone. For example, exchanging the order of two consecutive steps in the ID3 protocol (both of which returning a leaf node and then terminating the protocol) results in a protocol which cannot be simulated (see (31, page 13)).

4.3 Deterministic Encryption Reveals Information

A common misconception is that encrypting data, or hashing it, using any encryption system or hash function, keeps the data private. To show why this is not necessarily true, consider the following example, which illustrates an incorrect solution to the private matching problem of Section 3.3: Alice and Bob each have a list of k items, $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$, respectively. They wish to compute the intersection of their lists without revealing any other information. They use a deterministic hash function H which is believed to be collision intractable (and therefore suitable for cryptographic applications). Each party applies H to each of the k items in his or her list, and then they publish the resulting lists: $H(x_1), \ldots, H(x_k)$ and $H(y_1), \ldots, H(y_k)$. If a value occurs in both lists, they conclude that it corresponds to an item which appeared in the intersection of the two original lists. This solution indeed finds the intersection of the lists, but it might provide additional information if it is known that items in the list come from a relatively small domain. Bob can, for example, apply H to each possible value of x_1, and check whether the result is equal to the value $H(x_1)$, published by Alice. If this equality holds, Bob can deduce that he found x_1.

The problem exists if the domain is sufficiently small to enable one to exhaustively apply H to each item of the domain, or if the domain has limited min-entropy. For example, if Alice's items are known to be names of people, Bob can exhaustively apply H to every possible name, ordering his guesses according to the popularity of the names. We stress that although concrete attacks exist if the domain is (or may be) sufficiently small, the problem arises even for a large domain (unless one assumes that the function H is a random oracle). In particular, when a concrete hash function is applied to a large random value, it is still possible that *partial information* on the input is leaked, again revealing something that cannot be deduced in the ideal model.

The root of the problem is the use of a deterministic function (be it a hash function or

a deterministic encrypting scheme such as textbook RSA). One should therefore never apply a deterministic function to an item and publish the result. Instead, a semantically secure encryption scheme must be used. Unfortunately, this rules out a number of "simple and efficient" protocols that appear in the literature (indeed, these protocols are not and cannot be proven secure).

4.4 Security Proofs

It is tempting to prove security by stating what constitutes a "bad behavior" or an "illegitimate gain" by the adversary, and then proving that this behavior is impossible. Any other behavior or gain is considered benign and one need not bother with it. This approach is often easier than the use of simulation based proofs. The latter might also be considered overly cautious in preventing some far-fetched adversarial scenarios, whose gain to the adversary is unclear.

Consider, for example, a protocol in which Alice receives an encrypted message. We might assume that the only possible bad behavior is for her to try and decrypt the message, and that the only illegitimate gain she might obtain is learning information about the encrypted value. The use of a semantically secure encryption scheme should prevent this behavior. Assume, however, that Alice participates in an auction protocol and that the encrypted message contains another party's bid. Alice's goal might be to generate an encrypted bid which is only slightly higher than the other bid. The use of a semantically secure encryption system might not prevent this attack (especially if the encryption scheme is homomorphic).

It is hard to predict what type of corrupt behavior an adversary might take and thus dangerous to disregard any other behavior that we have not thought of as useless for the adversary. Indeed, real world attackers often act in ways which were not predicted by the designers of the system they attack. It is also hard to define what constitutes a legitimate gain by the adversary, and allow it, while preventing illegitimate or harmful gains. The notion of "harmful" might depend on a specific application or a specific scenario, and even then it might be very hard to define. We therefore urge protocol designers to prove security according to the simulation based definitions of Section 2, which prevent any attack which is not possible in an idealized scenario.

5 Secure Computation and Privacy-Preserving Data Mining

As we have seen, it is possible to securely compute *every* efficient functionality! Given this very strong result, it is tempting to state that the problems of privacy-preserving data mining are all solved in principle. Of course, it is still necessary to construct protocols that are efficient enough to be used in practice, but at least we know that "polynomial-time" solutions always exist. Unfortunately, this view of the role of secure computation in privacy-preserving data mining is far from accurate. The main reason for this is the following very important observation:

> *The field of secure multiparty computation deals with the question of how to securely compute a functionality, but does not ask the question of whether the functionality should be computed in the first place.*

Stated differently, secure multiparty computation tells us that for any functionality, it is possible to compute it without revealing anything beyond the output. However, it does not consider the

question of how much information about the input is revealed by that output. Take the example of computing the "average". A secure protocol can compute the average of parties' salaries without revealing anything beyond the output. However, if two parties run the protocol, then each party can compute the other party's salary exactly (given its own salary and the average). Thus, even though the protocol revealed nothing, the output itself reveals everything. This implies that although secure computation is an extraordinarily powerful tool and one that is very helpful in the field of privacy-preserving data mining, it can only be applied once it has been determined that the function in question is "safe" (i.e., the function output does not reveal sensitive information). This latter question—of what functions can be safely computed—is the focus of the field of "privacy". We stress that we do not belittle the role of secure computation in privacy-preserving data mining in any way. Rather, we see the fields of privacy and secure computation as complementary: the first is needed to decide that a given function is safe, and the second is needed in order to compute the function so that it remains safe (i.e., by using secure computation we are guaranteed that only the output is revealed and so the determination that the function is safe suffices for saying that it can be computed).

Privacy and statistics. Another area where privacy is crucial is that of public statistics. One classic case is that of the Census bureau that needs to publicize tables that sum up each census. This task is extremely dangerous because census questionnaires contain alot of sensitive information, and it is crucial that it not be possible to identify a single user in the publicized tables. In this area, it seems that secure computation is not much help. In particular, it seems that the function being computed is the tables, and there is no need to use any cryptography once the tables are deemed safe. Although this may seem obvious, we stress it because often it is suggested that the Census Bureau should run secure protocols with individuals and organizations who wish to carry out research on the census data, rather than have them simply release the tables. This suggestion is not helpful at all, in part due to the following two reasons: first, allowing citizens to obtain arbitrary statistics on the census data can be much more problematic than providing the data in the form of carefully prepared tables (maliciously prepared statistical queries can be used to target an individual citizen's data). Second, the Census bureau is a public body and as such must act in a transparent manner. The fact that some (or much) of the census results can be verified by comparing them to other studies means that the public has confidence in the accuracy of the census results. However, if secure protocols are run separately (in secret) between organizations seeking data and the Census bureau, this transparency is lost. Having said this, we strongly believe that the rigorous approach that is typical to cryptography will be instrumental in providing satisfactory solutions to the questions of privacy in this setting.

Applying secure computation. We conclude by remarking that even in cases where secure computation can be used, one must be careful in how it is applied. Specifically, we make the argument here that it is crucial to understand what the exact privacy concerns are when applying secure computation to privacy-preserving data mining problems. Consider an online shopping scenario in which a user's shopping habits are recorded. By applying data mining techniques, these records can be used to improve customer experience by offering products that are likely to be of interest. One example of the successful use of this technique is by `Amazon.com`, who offers products to its customers based on their previous purchases and searches (and on information gathered from other customers as well). If such a technique is used in a broad manner in which *all* of a user's buying habits are aggregated together, then this naturally raises privacy concerns. This is because what we consume says a lot about who we are. Thus, anyone with access to all of this information immediately knows a lot about our interests and

habits. Despite this, there can be considerable gain to the consumer by applying data mining techniques here. Targeted advertising, when done well, can bring to our attention products that we really are interested in purchasing and may therefore provide significant benefit. It therefore seems that we must choose between the desire to keep our personal information to ourselves and the desire to use that information to our benefit.

Given the background we have provided above in this paper, it is not hard to reach the conclusion that the ultimate solution is to run a secure computation protocol where each consumer holds their purchase history and personal interests, and each online store holds an algorithm that takes users' purchase histories and creates a personalized shopping catalog. Ignoring issues of efficiency, such a protocol seems to solve all problems. On the one hand, the consumer is provided with a personalized catalog that is based on his/her interests. On the other hand, the online store learns nothing about the consumer's private purchase history.

The problem with the above solution is that it does not address the real privacy concern that arises in the scenario of online shopping and users' purchase histories. In particular, although technology for personalizing shopping catalogs can be useful and positive, it can also be used for unfair price discrimination. For example, if a user's profile shows that they do not "shop around" and usually buy as soon as they find what they are interested in, then the shopping site may charge the user higher prices. Now, price discrimination is often considered a positive economic force as it enables sellers to charge more to those willing to pay more and less to others. However, in our example there is an inherent asymmetry: the seller has a lot of information about the buyer and what they are willing to pay; in contrast, the buyer does not have equivalent information about what price the seller is willing to sell for. To make things worse, the buyer is not even aware that the seller has their purchase profile and typically assumes that they are being charged the same amount as everyone else. This lack of symmetry between the buyer and seller creates an unfair disadvantage to the buyer. Notice now that the secure computation solution that we suggested above does *not* solve this problem! This is because the algorithm that the seller inputs to the protocol is part of its input, and so may determine rules for unfair price discrimination. Of course, the solution is very simple: the algorithm used by the seller must be public. It can thereby be scrutinized for elements of unfair price discrimination before a consumer agrees to use it. We are therefore not claiming that secure computation doesn't solve the problem. Rather, we are arguing that one needs to take care that the true concerns are addressed before implementing a solution. Fortunately, given the modeling of secure computation, it suffices to be convinced that the ideal-model functionality solves the privacy problem at hand (and one does not need to look at a complex protocol).

6 Future Challenges

Cryptographic protocols for secure computation achieve remarkable results: it has been shown that generic constructions can be used to compute any function securely, and it has also been demonstrated that some functions can be computed even more efficiently using specialized constructions. Still, a secure protocol for computing a certain function will always be more costly than a naive protocol that does not provide any security.

Cryptographers seek to make secure protocols as efficient as possible in order to minimize the performance gap between secure and naive protocols. Yet, another possible goal could be to examine the objective of secure computation. The current definitions of security provide a

very strong guarantee: minimal loss of information in the face of strong adversaries. As we have mentioned above, in some cases it makes sense to relax the definition of security in order to achieve security. We stress that this is *always* preferable to the approach of suggesting a highly efficient protocol that is not proven secure under any model. Such relaxations can come in many forms, from relaxing the adversary's power to allowing some leakage of information. We believe that further research in this area is crucial for the development of secure and efficient protocols in this field. Of course, this must go hand in hand with research on privacy in general and the question of what information leakage is acceptable and what is not.

References

[1] G. Aggarwal, N. Mishra and B. Pinkas. Secure Computation of the k-th Ranked Element. In *EUROCRYPT 2004*, Springer-Verlag (LNCS 3027), pages 40–55, 2004. 86, 87

[2] W. Aiello, Y. Ishai and O. Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. In *EUROCRYPT 2001*, Springer-Verlag (LNCS 2045), pages 119-135, 2001. 76

[3] Y. Aumann and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In *4th TCC*, Springer-Verlag (LNCS 4392), pages 137-156, 2007. 74, 75

[4] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991. 63

[5] D. Beaver, S. Micali and P. Rogaway. The Round Complexity of Secure Protocols. In *22nd STOC*, pages 503–513, 1990. 84

[6] M. Bellare and S. Micali. Non-Interactive Oblivious Transfer and Applications. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 547–557, 1989. 76

[7] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *20th STOC*, pages 1–10, 1988. 66, 67, 84

[8] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000. 63, 69, 72

[9] R. Canetti and A. Herzberg. Maintaining Security In The Presences Of Transient Faults. In *CRYPTO'94*, Springer-Verlag (LNCS 839), pages 425–438, 1994. 65

[10] D. Chaum, C. Crépeau and I. Damgård. Multiparty Unconditionally Secure Protocols. In *20th STOC*, pages 11–19, 1988. 66, 84

[11] R. Cleve. Limits on the Security of Coin Flips when Half the Processors are Faulty. In *18th STOC*, pages 364–369, 1986. 64

[12] R. Cramer, I. Damgård and J.B. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. In *EUROCRYPT 2001*, Springer-Verlag (LNCS 2045), pages 280–300, 2001. 84

[13] I. Damgård and M. Jurik. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *PKC 2001*, Springer-Verlag (LNCS 1992), pages 119–136, 2001. 79

[14] S. Even, O. Goldreich and A. Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, 28(6):637–647, 1985. 75, 76, 83

[15] R. Fagin, M. Naor and P. Winkler. Comparing Information Without Leaking It. *Communications of the ACM*, 39(5):77–85, 1996. 85

[16] M. Franklin and S. Haber, Joint Encryption and Message-Efficient Secure Computation. *Journal of Cryptology*, 9(4):217–232, 1996. 84

[17] M. Freedman, K. Nissim and B. Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT 2004*, Springer-Verlag (LNCS 3027), 1–19, 2004. 85, 86

[18] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In 19*th STOC*, pages 218–229, 1987. For details see (21, Chapter 7). 66, 67, 83, 84

[19] O. Goldreich. Cryptography and Cryptographic Protocols. In *Distributed Computing*, 16(2):177–199, 2003. 61

[20] O. Goldreich. *Foundations of Cryptography: Volume 1 – Basic Tools*. Cambridge University Press, 2001. 61

[21] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications*. Cambridge University Press, 2004. 61, 66, 67, 68, 69, 76, 83, 95

[22] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90*, Springer-Verlag (LNCS 537), pages 77–93, 1990. 63

[23] S. Goldwasser and S. Micali. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In 14*th STOC*, pages 365–377, 1982. 76

[24] S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988. 66

[25] S. Halevi and Y.T. Kalai. Smooth Projective Hashing and Two-Message Oblivious Transfer. *Cryptology ePrint Archive*, Report 2007/118. Preliminary version in *EUROCRYPT 2005*, Springer-Verlag (LNCS 3494), pages 78–95, 2005. 73, 76

[26] Y. Ishai, J. Kilian, K. Nissim and E. Petrank. Extending Oblivious Transfers Efficiently. in *CRYPTO 2003*, Springer-Verlag (LNCS 2729), pages 145–161, 2003. 76

[27] S. Jarecki and V. Shmatikov. Efficient Two-Party Secure Computation on Committed Inputs. In *EUROCRYPT 2007*, Springer-Verlag (LNCS 4515), pages 97–114, 2007. 83

[28] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC Press, 2007. 61, 76

[29] J. Kilian. *Founding Cryptography on Oblivious Transfer*. In 20*th STOC*, pages 20–31, 1988. 75

[30] L. Kissner and D. Song. Privacy-Preserving Set Operations. In *CRYPTO 2005*, Springer-Verlag (LNCS 3621), pages 241–257, 2007. 85

[31] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. *Journal of Cryptology*, 15(3):177–206, 2004. 88, 89, 90

[32] Y. Lindell and B. Pinkas. A Proof of Yao's Protocol for Secure Two-Party Computation. To appear in the *Journal of Cryptology*. Also appeared in the *Cryptology ePrint Archive,* Report 2004/175, 2004. 80, 81, 82

[33] Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries In *EUROCRYPT 2007*, Springer-Verlag (LNCS 4515), pages 52–78, 2007. 80, 82, 83

[34] D. Malkhi, N. Nisan, B. Pinkas and Y. Sella. Fairplay – A Secure Two-Party Computation System. In the 13*th USENIX Security Symposium*, pages 287–302, 2004. 83

[35] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991. 63

[36] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In the 12*th SIAM Symposium on Discrete Algorithms* (SODA), pages 448–457, 2001. 76, 77, 83

[37] M. Naor and B. Pinkas, Oblivious Polynomial Evaluation. *SIAM Journal on Computing*, 35(5):1254–1281, 2006. Extended abstract in 31*st STOC*, pages 245–254, 1999. 80, 85, 86

[38] M. Naor and O. Reingold. Number-Theoretic Construction of Efficient Pseudorandom Functions. In 38*th FOCS*, pages 458–467, 1997. 79

[39] J.B. Nielsen. Extending Oblivious Transfers Efficiently - How to get Robustness Almost for Free. *Cryptology ePrint Archive,* Report 2007/215, 2007. 76

[40] R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks. In 10*th PODC*, pages 51–59, 1991. 65

[41] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT'99*, Springer-Verlag (LNCS 1592), pages 223–238, 1999. 79

[42] M. O. Rabin. How to Exchange Secrets by Oblivious Transfer. Technical Memo TR-81, Aiken Computation Laboratory, 1981. 75

[43] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In 21*st STOC*, pages 73–85, 1989. 66

[44] M. Stadler. Publicly Verifiable Secret Sharing. In *EUROCRYPT'96*, Springer-Verlag (LNCS 1070), pages 190–199, 1996. 79

[45] A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986. 66, 67, 80

[46] HRDC Dismantles Longitudinal Labour Force File Databank. Press Release – Human Resources and Social Development Canada, May 29, 2000.
http://www.hrsdc.gc.ca/en/cs/comm/news/2000/000529_e.shtml 60

Acknowledgments

The research of the first author was supported by an Infrastructures grant from the Ministry of Science, Israel. The research of the second author was supported in part by the Israel Science Foundation (grant number 860/06).