14

Secure Multiparty Computation

14.1 Introduction **14**-1
14.2 What Is a "Secure" Protocol? **14**-2
Definition of Two-Party HBC Secure Protocols • Beyond Two-Party and HBC Adversaries
14.3 Survey of General SMC Results **14**-5
14.4 Methods for Practical Secure Protocols **14**-5
Splitting Values • Representative-Based Architectures for SMC
14.5 Logical Circuit Based Approaches **14**-7
Cryptographic Primitives • Simulating Circuits with Yao's Method • What about Malicious Behavior • Using Circuits
14.6 Computing on Encrypted Data **14**-11
Homomorphic Encryption • Scalar Product • Polynomial Operations • Set Operations
14.7 Composing Secure Protocols **14**-13
14.8 Summary **14**-14
References **14**-15

Keith B. Frikken
Miami University

14.1 Introduction

A plethora of mutually beneficial information collaboration opportunities exist when organizations and individuals share information. Some examples of such collaborations include (1) hospitals sharing information about patients with a rare disease in order to develop better treatments; (2) suppliers and retailers sharing their holding costs, inventory on hand, and other supply-chain information in order to reduce their costs; and (3) finding interorganization purchase patterns by sharing customer records to perform collaborative data mining. Clearly, there are many potential benefits, both monetary and societal, for such collaborations, however, one significant drawback is concerns about confidentiality and privacy that are associated with sharing information. Returning to the examples above, sensitivity concerns include (1) patients' information is private and there may be laws preventing such sharing of information (e.g., in the United States, HIPAA restricts the sharing of medical information); (2) the supplier and retailer may be concerned that their cost information could be used against them in a future interaction; and (3) sharing customers' information may violate the customers' privacy. Thus, while significant benefit may result from such collaborations, these collaborations may not occur due to privacy and confidentiality concerns.

Secure protocols (also called privacy-preserving or confidentiality-preserving protocols) are cryptographic techniques that obtain the result of information collaboration while preserving privacy and confidentiality. More specifically, secure protocols are cryptographic techniques for computing functions (i.e., collaboration outcomes) over distributed inputs while revealing only the result of

the function. Thus, secure protocols are a way to have the best of both worlds—we can obtain the benefit of collaboration without the drawback of revealing too much information. While this may seem like an intractable goal for many problems, there have been many general results which state, under various assumptions, that any function computable in polynomial time can also be computed securely with polynomial communication and polynomial computation. The problem of computing any function in a secure manner is called secure multiparty computation (SMC).

This chapter provides a description of techniques for building secure protocols from an applied standpoint. That is, this chapter is not an exhaustive description of SMC techniques, but rather this chapter describes a few key building blocks that facilitate the creation of several secure protocols. For the reader that is interested in a formal description of cryptographic primitives and SMC we refer you to [15,16]. The rest of this chapter is organized as follows: in Section 14.2 secure protocols are defined more formally. Section 14.3 presents a survey of known results for secure protocols. Section 14.4 describes practicality problems for the general SMC results along with methods that help overcome these problems. In Sections 14.5 through 14.7 several specific techniques for two-party secure protocols are described, and in Section 14.8 we summarize this chapter.

14.2 What Is a "Secure" Protocol?

Before describing techniques for "secure" protocols, one must define what is meant by "secure." Before formally defining this notion, we provide some intuition into the definition. One trivial way to securely compute any function is for all parties to send their input to a fully trusted third party (TTP), and this TTP then computes the desired result and sends the output of the function to each party. Obviously, it is unlikely that such a TTP exists, but this protocol achieves the best possible outcome—that is, the only information revealed about inputs is the output of the protocol and inferences that can be made from this output along with one or more inputs.* Thus a protocol is considered "secure" if it is no worse than the above-described TTP protocol. More specifically, a secure protocol is a protocol that reveals only the result of the function and inferences that can be deduced from this output with one or more input values.

The following is a simple example of a secure protocol that helps clarify the above intuition. Suppose N people are sitting around a table lamenting that they are vastly underpaid; to help justify their complaints, this group has a genuine interest in finding the mean salary of everyone at the table, but due to privacy concerns, they do not want to reveal individual salaries to the group. Now if there are only two people at the table, then when given the result (i.e., the average salary) and one person's salary, one can deduce the other person's salary. Thus for $N = 2$ a secure protocol is as simple as having both parties reveal their salaries to each other, because one party's value can be deduced from the result and the other input. However, when $N \geq 3$, such exact inferences cannot be made, so a more complicated protocol is needed.

One way to compute the average salary (for $N \geq 3$) is as follows. The first person chooses a random number R from a range of values much larger than the range of salaries. This person adds his salary to R, writes the result on a slip of paper, and then passes this piece of paper to the next person at the table. When this person receives the slip of paper, he adds his salary to the value on the paper, writes down the new sum on a different piece of paper, destroys the original piece of paper, and passes the new paper to the person next to him. The paper passing continues until everyone has incorporated their salary into the sum. After everyone has had a turn, the final piece of paper is passed back to the first person. The first person then subtracts R from the sum to learn the sum of everyone's salary. He then divides this sum by the number of people to learn the average salary and then he reveals the average salary to everyone at the table.

* In a multiple party protocol it is possible that participants will collude and share inputs with each other.

Let us examine the properties of this average salary protocol. By adding a large random value R to the first person's salary, this person's salary is hidden with high probability (furthermore the probability is controllable by the range from which R is chosen). More generally, assuming that there is no collusion between members, then all that the ith person learns from the protocol is the sum of the first $i-1$ salaries and R, and since R is chosen from a large range, the sum of the first $i-1$ salaries is hidden with high probability. Of course, there are some problems with this protocol: (1) the protocol is not collusion-resistant—for example, if the 2nd and 4th parties collaborate together, then they could learn the 3rd party's salary; (2) the first person can learn the actual result and announce a different result; and (3) any participant can modify the value that is being passed around the group to increase or reduce the average.

Clearly, there are different types of adversaries that must be considered when creating secure protocols. There are two basic types of adversaries that are considered in the SMC literature. The first adversary type is called an honest-but-curious (HBC) adversary model (also called semi-honest or passive adversaries). In this adversary model, the adversary will faithfully follow the specified protocol, but after the protocol has completed it will attempt to learn additional information about other inputs. While this model is unrealistic in many regards, if an efficient protocol cannot be developed for this simplified model, then there is little hope that a protocol can be developed for a more realistic adversary model. The second adversary model is a more realistic adversary model, and it is often called the malicious or active adversary model. A malicious adversary will deviate arbitrarily* from the protocol in order to gain some advantage over the other participants. The advantages that an adversary can try to obtain include (1) to learn additional information about other inputs; (2) to modify the result of the protocol; (3) to abort the protocol prematurely (perhaps after the adversary learns a partial result it aborts the protocol to prevent others from learning the results).

14.2.1 Definition of Two-Party HBC Secure Protocols

We are now ready to formalize the above informal definition for two parties in the HBC adversary model. Recall that a two-party protocol is secure if everything that can be computed from the protocol can be deduced from the output and one of the input values. To make this more concrete suppose Alice and Bob are engaging in a protocol to compute a function f. This protocol involves one or more message exchanges between Alice and Bob; we will call these messages the transcript of the protocol. It must be shown that the transcript sent from Alice and Bob (and vica versa) does not reveal too much information. More specifically, the transcript sent from Alice to Bob should be computable from the result of the protocol and Bob's inputs, otherwise the protocol would reveal more information to Bob than the results (a similar statement must be made about the transcript from Bob to Alice). In the remainder of this section we formalize this definition. The reader that is already familiar with the cryptographic literature should feel free to skip the remainder of this section, and we refer the reader that is interested in more details about these definitions to [15,16].

As any cryptographic protocol will fail with some probability (one can always guess the other party's inputs), it is necessary to define an acceptable probability threshold for which a protocol is deemed secure. Clearly, a fixed constant probability is undesirable since an adversary can simply repeat the attack several times in order to increase the probability. The standard technique used in the cryptographic literature is the notion of a *negligible* probability. More specifically, a function $\mu(k)$ is negligible in k if for any polynomial p, for large enough k, $\mu(k) < \frac{1}{p(k)}$. A protocol is deemed secure if the adversary's success probability is negligible in a security parameter. One advantage of

* Adversaries are usually modeled as probabilistic-polynomial time adversaries, and this is the only bound placed on malicious adversaries.

using a negligible probability is that even if the adversary repeats the attack a polynomial number of times, the success probability will still be negligible.

As mentioned above, to show that a protocol is secure, it must be shown that the communication transcript from the protocol can be simulated when given the results and one party's input. One way to do this is to require that the communication transcript be exactly generatable from the output of the protocol, but this is very limiting. However, if the adversary is computationally-bounded, then it is enough to generate a transcript that is so close to the real transcript that the adversary cannot distinguish between the real and simulated transcripts. In the cryptographic literature, this notion of "close enough" is called computational indistinguishability. More formally, two random probability ensembles $\{X_n\}_{n\in N}$ and $\{Y_n\}_{n\in N}$ are *computationally indistinguishable* if for any probabilistic polynomial time algorithm D the following value is negligible in n:

$$|Pr[D(X_n, 1^n) = 1] - Pr[D(Y_n, 1^n) = 1]|$$

It is now possible to define a secure two-party protocol in the HBC model. Suppose that Alice and Bob have respective inputs x_A and x_B, and that at the end of the protocol Alice should learn $f_A(x_A, x_B)$ and Bob should learn $f_B(x_A, x_B)$. Note that in many cases f_A and f_B will be the same function, but to make the definition as general as possible, it incorporates the possibility that Alice and Bob will learn different results. Now given a protocol Π, Alice's view of the protocol (i.e., all messages she receives from Bob) is denoted by $\{view_A^{\Pi}(x_A, x_B)\}_{x_A,x_B\in\{0,1\}^*}$, and similarly Bob's view is denoted by $\{view_B^{\Pi}(x_A, x_B)\}_{x_A,x_B\in\{0,1\}^*}$. The protocol Π is said to be secure in the HBC model if there exist probabilistic polynomial time simulation algorithms S_A and S_B, such that (1) $\{S_A(x_A, f_A(x_A, x_B))\}_{x_A,x_B\in\{0,1\}^*}$ and $\{view_A^{\Pi}(x_A, x_B)\}_{x_A,x_B\in\{0,1\}^*}$ are computationally indistinguishable and (2) $\{S_B(x_B, f_B(x_A, x_B))\}_{x_A,x_B\in\{0,1\}^*}$ and $\{view_B^{\Pi}(x_A, x_B)\}_{x_A,x_B\in\{0,1\}^*}$ are computationally indistinguishable. To see how this definition matches the intuition from before, this definition states that an efficient algorithm exists that can simulate all of the messages sent by Bob (resp. Alice) when given only Alice's (resp. Bob's) input and output. Thus anything learned from the protocol must also be learnable from the result alone, and therefore the protocol reveals only the result and inferences derived from this result.

14.2.2 Beyond Two-Party and HBC Adversaries

The definition in the previous section can be extended to the malicious adversary model, but as this definition is quite cumbersome we opt for an informal discussion (we refer the interested reader to [16] for more details). Reconsider the TTP setting for computing a function securely, while this protocol is unrealistic it does provide an "ideal" goal for an SMC protocol. That is, a protocol will be secure if for any attack in the real protocol, there is another attack (with similar adversary complexity) in the ideal protocol with the TTP. Consider a malicious adversary in the TTP setting, this adversary can do the following actions: (1) refuse to participate in the protocol; (2) abort the protocol prematurely; and (3) modify its inputs. A protocol is secure in the malicious model if for any PPT adversary in the real protocol there is a malicious adversary in the TTP setting that can achieve the same* results. One limitation of the two-party secure computation is fairness of the protocols, that is, one party will learn its result before the other party and then can stop participating in the protocol to prevent the other party from obtaining its output. Unfortunately, the impossibility results in Byzantine agreement [11] imply that fairness is not achievable in two-party computation (e.g., a strict majority of the parties must be honest).

* That is, computationally indistinguishable.

14.3 Survey of General SMC Results

The first protocol for computing any two-party function securely in the semi-honest model was proposed in [30], and was later improved in [31] (in the latter we refer to this scheme as Yao's scheme), to require only a constant number of communication rounds. The basic idea of this approach is to build a logical circuit for the function in question, and then to use a secure protocol to blindly evaluate the circuit (for details on how this is accomplished, see Section 14.5). While the original scheme did not have a detailed security proof, the scheme was proven secure in [21]. Furthermore, Yao's scheme has been implemented in the FAIRPLAY system [22], and it was shown that this scheme is practical for some problems. Several protocols have extended Yao's system to make it secure in the malicious model [22,24,29]).

With regards to multiparty computation, it was shown in [17] that as long as a majority of the participants are honest, it is possible to securely compute any function. In [23] a scheme was given to securely compute any function in such a model in a constant number of rounds. There have been several extensions to more sophisticated adversary models, including (but not limited to) adaptive adversaries that corrupt participants after the protocol has began [5] and secure protocols against computationally unbounded adversaries [2,6].

14.4 Methods for Practical Secure Protocols

In this section, we consider techniques for making secure protocols practical. Since the general results of SMC state that there is a secure protocol for any number of participants, a first approach is to have all participants engage in a secure protocol as depicted in Figure 14.1. Typically, such schemes require a portion (e.g., a strict majority) of the participants to be honest. These protocols are robust in that after the initial rounds of such protocols, even when some of the participants abort the protocol (intentionally or unintentionally) the rest of the participants can still compute the result. While this natural protocol architecture does allow for secure computation there are several disadvantages including (1) all participants must be online at the same time which may be difficult in some environments and (2) for large numbers of participants these protocols are very expensive. In summary, this architecture works very well for a small number of participants (especially for two people), but it does not scale well.

To further emphasize the problems with this initial architecture consider creating a secure protocol for an auction system, where several bidders are bidding on an item held by a seller. Clearly, bidders will want to protect their bid values. In this architecture all of the bidders and the seller would have to agree upon a fixed time to engage in a protocol to compute the results. In many auctions this is impractical, because the identities of the bidders is unknown until they make a bid, and furthermore the number of bidders could make this protocol impractical.

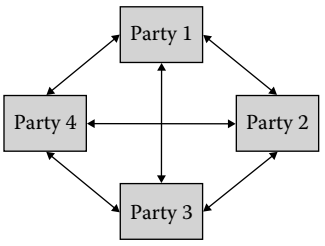

FIGURE 14.1 Example of first architecture.

Thus, for many problems having all participants engage in a protocol together is unlikely to be practical. One way to mitigate this problem is for many of the participants to delegate their portion of the protocol to another party. Of course, participants would not want to reveal their values to their representatives, but if the inputs could be split among different representatives so that no small group of the representatives can learn the values, then this may be acceptable. In the next section we describe various techniques for splitting values and then in Section 14.4.2 we formalize this representative-based architecture.

14.4.1 Splitting Values

The following are three techniques for splitting a value x among n participants.

Sharing: In sharing approaches all n participants are required to recover the value. One example, of this approach is that party i has a value x_i such that $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ where $\oplus$ is XOR. To split x in such a manner, $n-1$ random values are chosen for $x_1, \ldots, x_{n-1}$ and x_n is set to $x \oplus x_1 \oplus \cdots \oplus x_{n-1}$. Another similar sharing approach is to store x as the sum of n values modulo some value. That is, the shares of a value x are $x_1, \ldots, x_n$ where $x = x_1 + x_2 + \cdots + x_n \bmod N$ for some value N.

When given values in a modularly additive split format, then one can perform certain arithmetic operations on the split values. For example, if two values x and y are split among a group of participants then without communicating this group can compute $x + y$ (modulo the splitting modulus) by having each member add up their shares of x and y. Also, a group of participants can multiply their shared value by a constant by each multiplying their individual values by the constant.

Threshold sharing: Another approach to sharing values is to split the values in such a way that t participants are required to recover the value ($t < n$). A classic technique for achieving such sharing is Shamir's secret sharing [28]. In this approach a $t-1$ degree polynomial P is chosen such that $P(0) = x$. Furthermore, participant i is given $(i, P(i))$. Now if any t parties share their values, they can interpolate their values to recover P and thus learn $P(0)$ (i.e., x). However, $t-1$ or less participants do not have enough information to reconstruct P and thus x is still protected. This is often referred to as a (t, n) threshold sharing scheme.

When given values split using such a sharing approach, it is possible to perform some mathematical operations on the values without having the participants communicate. Suppose that values x and y are split with a (t, n) sharing scheme and where each participant's points have the same x-coordinate. Then it is possible to compute $x + y$ in a threshold (t, n) simply by having each participant add the y-coordinates of their points. Furthermore, it is also possible to compute $x * y$ in a $(2t, n)$ shared manner by having each participant multiply the y-coordinates of their points. Finally, it is trivial to multiply a (t, n) shared value by a constant and obtain a (t, n) shared value by having each participant multiply their y-coordinate by the constant.

Encoding/value: Suppose that $x \in \{0, 1\}$ and that $n = 2$. In this case a specialized form of splitting is achieved by having one party learn two large random values v_0 and v_1 and having the other party learn v_i. Thus the first party knows the semantics of the encoding, but does not know the actual value. On the other hand, the second party knows the encoded value but does not know what it means. To split a value in this form the party owning x chooses v_0 and v_1 and sends them to party one and then sends v_x to party two. Of course, this can be extended to ℓ values in a larger range in a natural way. Furthermore, this idea can be extended to multiple parties by splitting the encodings among the parties (using either sharing or threshold sharing) and giving all parties the value.

14.4.2 Representative-Based Architectures for SMC

An architecture (formalized in [9]) that increases scalability is to separate the participants into three, not necessarily mutually exclusive, groups: input servers, computation servers, and output servers. The input servers split their inputs (using the techniques described in the previous section) among the computation servers so that no small group of computation servers can recover the inputs. The computation servers then engage in a secure protocol to compute the results in a split fashion. And finally, the split results are sent to the output servers who then learn the results. This representative-based architecture is depicted in Figure 14.2. This architecture mitigates many of the disadvantages of the original protocol architecture described earlier. That is, the input servers do not all need to be online at the same time because they can submit their values to the computation servers and then go offline. Furthermore, the number of computation servers is typically much smaller than the number

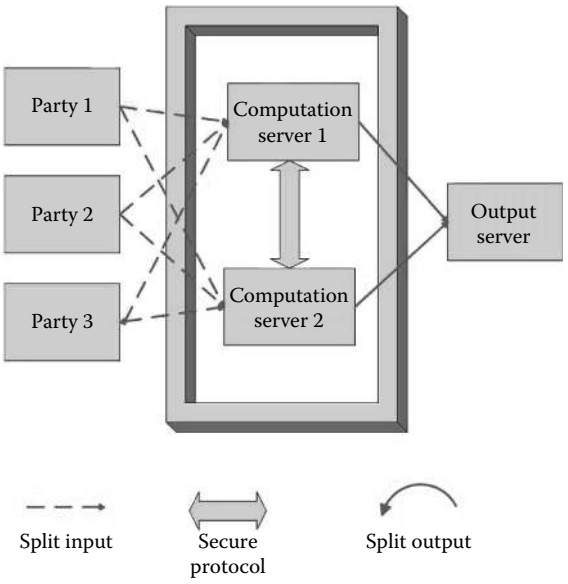

FIGURE 14.2 Example of representative-based architecture.

of input servers, and thus this approach is more scalable. The downside of this representative-based approach is that this usually requires some level of trust in an outside party (i.e., a party that is neither an input nor an output server).

In [25] a version of this representative-based architecture was used to provide a practical privacy-preserving auction. In this scheme a third party called the auctioneer is utilized. The sole purpose of the auctioneer is to help the seller compute the winning bid. The bidders split their bids between the auctioneer and the seller so that neither individual learns information about the bids. Once the auctioneer and the seller have received all bids, the seller and the auctioneer engage in a secure protocol to compute the winning bidder along with the winning bid in a split fashion. This information is then revealed to the seller and to the bidders. The only trust assumption is that the bidders trust the auctioneer not to collude with the seller, and this level of trust is more reasonable than the fully trusted third party approach, and may be applicable in many practical situations.

Since the focus of this chapter is practical SMC techniques, the focus will be on two-party SMC techniques (as these are more likely to be practical). Thus for problems with more than two parties we will assume a representative-based architecture with two computational servers.

14.5 Logical Circuit Based Approaches

In this section we discuss a technique for computing any two-party function in a secure manner. The main focus is on the HBC model, but extensions to the malicious model are also discussed. The principal idea is that we will represent the function f as a logical circuit C_f. In [31], a technique was described that securely evaluates a logical circuit with communication and computation proportional to the number of gates in the circuit and with a constant number of rounds; in [21] this technique was proven secure. Now, any function computable in polynomial time can be computed with a polynomially sized logical circuit, and so these two things imply that any function computable in

polynomial-time can be computed securely with polynomial communication and computation and a constant number of rounds. In the remainder of this section we first describe the cryptographic primitives needed for Yao's construction in Section 14.5.1. Then Yao's scrambled circuit evaluation technique for the HBC adversary model is described in Section 14.5.2. In Section 14.5.3 extensions to the malicious model are described. Finally, in Section 14.5.4 applications of Yao's approach are described.

14.5.1 Cryptographic Primitives

One building block that is used is a form of symmetric key encryption. The encryption scheme for Yao's scrambled circuit evaluation requires specific properties: elusive range (it is difficult to choose a value that is a valid ciphertext for a particular key k) and efficiently verifiable range (given a ciphertext and a key it is possible to determine if the ciphertext is a possible encryption with the key k). These properties are possible for cryptosystems, and we refer the reader to [21] for more information about these properties. In the remainder of the paper we use the notation $Enc(M, k)$ to denote the encryption of the message M with the key k.

The other core building block needed for Yao's scrambled circuit evaluation is 1-out-of-2 chosen oblivious transfer; oblivious transfer (OT) was introduced in [27]. In the original OT protocols the sender would have a message and the receiver would obtain the value with probability one-half. In [10], a variation was introduced where the sender has two messages and the receiver obtains message one with probability one-half and receives message two with probability one-half; furthermore, the sender would not know which message the sender obtained. While these two versions of OT seem different, they were proved to be equivalent in [7]. The version that will be used in the remainder of the chapter is chosen 1-out-of-k OT (also called all or nothing disclosure of secrets) and was introduced in [3]. In this version of OT the sender has k messages and the receiver obtains a specific message that the receiver gets to choose. We will denote this protocol as $OT_1^k((M_1, M_2, \ldots, M_k), i)$ where the receiver learns M_i and the sender learns nothing.

The following is a protocol described in [1] for chosen 1-out-of-2 OT. Note that there are many other protocols for OT in the literature, but as OT is not the focus of this chapter only one such protocol is described (see Figure 14.3).

The above protocol for OT requires a single round (once the setup has been done), and it requires $O(1)$ modular exponentiations and other computations. We now give a sketch of security argument for the above protocol. First, an honest receiver will obtain M_b, because $\beta_b^r = (g^{s_b})^r = (g^r)^{s_b} = \gamma_b$. However, the receiver cannot obtain M_{1-b}, because this would imply that the receiver also learns

Input: The sender inputs two messages M_0 and M_1, and the receiver inputs $b \in \{0, 1\}$.
Output: The receiver obtains M_b.

1. The sender chooses a large prime p, a generator g of Z_p^*, and a value C in Z_p^* where the receiver does not know the discrete log of C. The values p, g, and C are sent to the receiver. Note that this step needs to be done only once for several OT protocols.
2. The receiver chooses a random value r computes two values α_0 and α_1 where $\alpha_b = g^r$ and $\alpha_{1-b} = \frac{C}{g^r}$. The receiver sends α_0 to the sender.
3. The sender computes $\alpha_1 = \frac{C}{\alpha_0}$. It then chooses two random values s_0 and s_1, and computes: $\beta_i = g^{s_i}$, $\gamma_i = \alpha_i^{s_i}$, and $\delta_i = M_i \oplus \gamma_i$ for $i \in \{0, 1\}$. It then sends β_0, β_1, δ_0, and δ_1 to the receiver.
4. The receiver computes $\beta_b^r = (g^{s_b})^r = (g^r)^{s_b} = \gamma_b$. The receiver then computes $\delta_b \oplus \gamma_b = M_b$.

FIGURE 14.3 Oblivious transfer protocol.

γ_{1-b}, which is $\left(\frac{C}{g^r}\right)^{s_{1-b}}$, which is $g^{ms_{1-b}}$ for some value m. The receiver does not know this value, because otherwise the receiver would know the discrete logarithm of C. Thus the receiver knows g^m and $g^{s_{1-b}}$, and needs to calculate $g^{ms_{1-b}}$ (which is the Diffie–Hellman problem). The sender is unaware of which item the receiver is choosing because $\frac{C}{g^r}$ and g^r are indistinguishable for a randomly chosen (and unknown) value r. For a more detailed argument as to why this scheme is secure see [1].

14.5.2 Simulating Circuits with Yao's Method

A high-level overview of Yao's approach is as follows: one party is labeled the circuit generator and the other party is labeled the circuit evaluator. For each wire in the circuit the generator creates two encodings (one for 0 and one for 1), and the evaluator will learn the encoding that corresponds to the actual value of each wire without knowing what the encoding corresponds to. A crucial piece of this protocol is a gate encoding that allows the evaluator to obtain the gate's output wire's encoding when given the gate's input wires' encodings. Finally, to learn the final result the evaluator is given the mapping between the circuit's output wire encodings and their values. In what follows we describe the details of this process.

Setup: Assume that Alice is playing the role of the generator and that Bob is the evaluator. Suppose the circuit in question consists of wires $w_1, \ldots, w_n$ that are partitioned into four mutually exclusive sets A (Alice's inputs), B (Bob's inputs), I (intermediate wires), and O (output wires). Furthermore, suppose these wires are connected to a set of binary gates $G_1, \ldots, G_m$ and that each wire is either in $A \cup B$ or is an output from some gate. We denote the actual value of the wire w_i by v_i, and we denote the binary function that corresponds to gate G_i ad g_i, e.g., $g_i : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$. Finally, the input wires to G_i are denoted by x_i and y_i, and the output wire is denoted by z_i. That is, $v_{z_i} = g(v_{x_i}, v_{y_i})$.

Circuit generation: For each wire w_i Alice randomly chooses two encryption keys $w_i[0]$ and $w_i[1]$, and she also chooses a random bit $\lambda_i \in \{0,1\}$. One of the goals of the protocol is for Bob to learn $(v_i \oplus \lambda_i)||w_i[v_i \oplus \lambda_i]$ for each wire in the circuit. Note that the λ value hides from Bob the actual value of a wire.

Furthermore, for each gate G_i, the generator creates four messages, denoted by $G_i[0,0]$, $G_i[0,1]$, $G_i[1,0]$, and $G_i[1,1]$, where $G_i[a,b]$ is $Enc(s_i[a,b]||w_{z_i}[s_i[a,b]], w_{x_i}[a] \oplus w_{y_i}[b])$* where $s_i[a,b] = g_i(a \oplus \lambda_{x_i}, b \oplus \lambda_{y_i}) \oplus \lambda_{z_i}$.

Circuit evaluation: The circuit generator and the evaluator do the following:

1. For each $a \in A$, Alice sends $(v_a \oplus \lambda_a)||w_a[v_a \oplus \lambda_a]$ to Bob.
2. For each $b \in B$, Alice and Bob engage in $OT_1^2(\{\lambda_b||w_b[\lambda_b], \bar{\lambda_b}||w_b[\bar{\lambda_b}]\}; v_b)$ where Alice plays the role of the sender.
3. Alice sends to Bob $G_i[0,0]$, $G_i[0,1]$, $G_i[1,0]$, and $G_i[1,1]$ for every gate.
4. For each $o \in O$, Alice sends λ_o to Bob.

After Bob receives the above information, he evaluates the circuit. First, for each wire $i \in A \cup B$, Bob knows $(v_i \oplus \lambda_i)||w_i[v_i \oplus \lambda_i]$ (from Steps 1 and 2 in the protocol above). Once Bob has learned the encodings for wires x_j and y_j, he computes $Dec(G_j[v_{x_j} \oplus \lambda_{v_{x_j}}, v_{y_j} \oplus \lambda_{v_{y_j}}], w_{x_j}[v_{x_j} \oplus \lambda_{v_{x_j}}] \oplus w_{y_j}[v_{y_j} \oplus \lambda_{v_{y_j}}])$

* Note that this notation indicates that $s_i[a,b]||w_{z_i}[s_i[a,b]]$ is encrypted with the key $w_{x_i}[a] \oplus w_{y_i}[b]$, and thus to be able to decrypt this value one would need both $w_{x_i}[a]$ and $w_{y_i}[b]$.

to obtain the encoding of the output wire of G_j (i.e., wire z_j). That is, this value is $(v_{z_j} \oplus \lambda_{v_{z_j}})||w_{z_j}[v_{z_j} \oplus \lambda_{v_{z_j}}]$. Once Bob has these values for all wires, he can compute v_o for all wires in O, since he knows λ_o (from Step 4 of the protocol above).

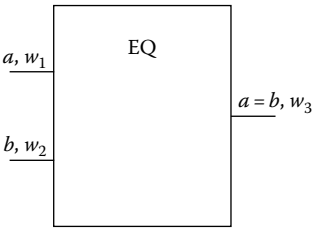

FIGURE 14.4 Example circuit.

To clarify the details of the above protocol, consider the following example. Suppose we have a small circuit where Alice has a single input bit a and Bob has a single input b. Furthermore, the output of the circuit is the predicate $a = b$. Note that this can be viewed as a circuit with a single equality gate (i.e., a not XOR gate). Furthermore, we will label the circuit as in Figure 14.4. To make this a concrete example, let us suppose that $a = 0$ and $b = 1$.

For each wire w_i Alice will choose encoding values $w_i[0]$ and $w_i[1]$ and λ_i. Let us suppose that Alice chooses $\lambda_1 = 0$, $\lambda_2 = 1$, and $\lambda_3 = 1$. Alice will send to Bob the values $(a \oplus \lambda_1)||w_1[a \oplus \lambda_1]$, that is she sends $0||w_1[0]$ to Bob. Alice and Bob will also engage in a 1-out-of-2 OT to reveal $0||w_2[0]$ (this corresponds to Bob's input wire).

For the gate information, Alice will calculate the following four messages:

- $s_1[0,0] = EQ(\lambda_1, \lambda_2) \oplus \lambda_3 = 1$
- $s_1[0,1] = EQ(\lambda_1, \bar{\lambda_2}) \oplus \lambda_3 = 0$
- $s_1[1,0] = EQ(\bar{\lambda_1}, \lambda_2) \oplus \lambda_3 = 0$
- $s_1[1,1] = EQ(\bar{\lambda_1}, \bar{\lambda_2}) \oplus \lambda_3 = 1$

Thus the gate information that Alice uses will be as follows:

- $G_1[0,0] = Enc(1||w_3[1], w_1[0] \oplus w_2[0])$
- $G_1[0,1] = Enc(0||w_3[0], w_1[0] \oplus w_2[1])$
- $G_1[1,0] = Enc(0||w_3[0], w_1[1] \oplus w_2[0])$
- $G_1[1,1] = Enc(1||w_3[1], w_1[1] \oplus w_2[1])$

Once Bob has this gate information, and he has the values $0||w_1[0]$ and $0||w_2[0]$ the only gate message he can decrypt is $G_1[0,0]$, and he thus receives $1||w_3[1]$. If this was to be used as an intermediate wire, he would not know the actual value of wire w_3, because it depends on λ_3 which he does not know. However, if this is an output wire, then Alice will reveal λ_3 and he will learn that the result is actually 0.

14.5.3 What about Malicious Behavior

Consider a malicious adversary in Yao's protocol. If the malicious player is the evaluator, then there is little that this player can do other than change his inputs. However, if the generator is malicious then this adversary can create any circuit with the same topology as the desired function. One mechanism that has been suggested to mitigate this attack is a cut-and-choose approach [22,24,25,29]. In this approach the generator creates several versions of the circuit and sends them all to the evaluator. The evaluator then requests that the generator "open" (i.e., give them all wire keys) a subset of the circuits. The evaluator then verifies that this subset of circuits was created properly.

In the simplest instantiation of this scheme the generator creates N circuits, and the evaluator verifies $N-1$ of the circuits. In this case, if a malicious generator wants to evaluate a faulty circuit, the malicious generator's chances of not being caught is $\frac{1}{N}$. In a more complicated approach the evaluator verifies $\frac{N}{2}$ circuits and computes the result for the other $\frac{N}{2}$ circuits. If the evaluated circuits' outputs differ then the evaluator sets the result to the most frequent output. In this case a cheating adversary is successful with a probability that is exponentially small in N. Many of the details on how this

cut-and-choose model is implemented have been omitted, but we refer the reader to [22,24,25,29] for more details.

14.5.4 Using Circuits

To use the scrambled circuit evaluation approach one needs to construct a circuit for the problem at hand. One approach that was used in FAIRPLAY [22] was to build a compiler that converted a programming language into a circuit. However, depending on the needs of the application one may decide to construct the circuits by hand. Some common circuits are described below.

- To test two n-bit values, $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ for equality one can use the following circuit: $\bigwedge_{i=1}^{n} \overline{(a_i \oplus b_i)}$. Note that this circuit requires $O(n)$ gates.
- To test if an n-bit value $a_1, \ldots, a_n$ is greater than another n-bit value $b_1, \ldots, b_n$ one can use the following circuit: $\bigvee_{i=1}^{n}(a_i \wedge \bar{b}_i \wedge \bigwedge_{j=1}^{i-1}(\overline{(a_j \oplus b_j)}))$. Note that this circuit has $O(n)$ gates.

One difficulty with using this approach is the difficulty with constructing circuit for some problems. Furthermore, circuits are inefficient for some problems such as indirect access into a list of values, that is if there is a list of items $v_1, \ldots, v_n$ given and an index $i \in [1, n]$ is computed by the circuit, then it requires $O(n)$ computation to obtain v_i in the circuit.

14.6 Computing on Encrypted Data

Because of the results in the previous section, one may wonder if all secure protocol problems have been solved. While the above techniques do imply that any polynomially computable function can be securely evaluated in the semi-honest model with polynomial communication, it is believed that for many problems the general solutions may not be practical. However, in some situations an efficient domain-specific protocol exists for some problems [18]. In this section we focus on one class of problems that have solutions based on arithmetic expressions.

14.6.1 Homomorphic Encryption

A homomorphic encryption scheme allows computation using encrypted values; this is useful because it facilitates some protocols based on arithmetic that are more efficient than their circuit counterparts. The idea behind these types of protocols is that the values can be encrypted and then someone can compute the encryption of a result without knowing the values. Specific homomorphic encryption schemes are described in [8,26]. In [26] the arithmetic is done mod n where n is an RSA modulus, and in [8], the arithmetic is done mod n^k where n is an RSA modulus and k is an integer. We now formally describe the properties of a homomorphic encryption scheme.

1. *Public key:* The systems are public-key encryption schemes in that anyone with the public parameters of the scheme can encrypt, but only those with the private parameters can decrypt.
2. *Semantically secure:* We require that the scheme be semantically-secure as defined in [19]. That is, given the following game between a probabilistic polynomial time (PPT) adversary A and a challenger C:
   a. C creates a public–private key pair (E, D) for the encryption system and sends E to A.
   b. A generates two messages M_0 and M_1 and sends them back to C.

c. C picks a random bit $b \in \{0, 1\}$ and sends $E(M_b)$ to A.

d. A outputs a guess $b' \in \{0, 1\}$.

We say that A wins the game if $b' = b$. We define the advantage of A for a security paramater k to be $Adv_A(k) = Pr[(b = b') - (1/2)]$. We say that the scheme is semantically secure if $Adv_A(k)$ is negligible in k.

3. *Additive:* Given the $E(x)$ and $E(y)$ and the public parameters of the scheme, one can compute $E(x + y)$ by calculating $E(x) * E(y)$.
4. *Multiplication:* A natural extension of the previous property is that when given $E(x)$ and c, one can compute $E(xc)$ by calculating $E(x)^c$.
5. *Re-encryption:* When given an encryption $E(x)$, one can compute another encryption of x, simply by multiplying the original encryption by $E(0)$.

14.6.2 Scalar Product

It should not come as a surprise that homomorphic encryption facilitates efficient protocols that compute some arithmetic expression. One example of such a protocol is the calculation of the scalar product of two vectors. That is given $\vec{A} =< a_1, \ldots, a_n >$ and $\vec{B} =< b_1, \ldots, b_n >$, the goal is to compute $\vec{A} \circ \vec{B} = \sum_{i=1}^{n} (a_i * b_i)$. One protocol for secure scalar product was introduced in [14], and Figure 14.5 is such a protocol.

14.6.3 Polynomial Operations

Another application of homomorphic encryption is the ability to compute various polynomial operations. This has been useful for set operations (which are described in detail in the next section), including [12,13,20]. To encrypt a polynomial with homomorphic encryption, each coefficient of the polynomial is encrypted with the encryption scheme. That is, the encryption of $P(x) = p_n x^n + \cdots + p_1 x + p_0$ is $E(p_n), \ldots, E(p_1), E(p_0)$, and we denote this value by $E_{poly}(P)$. Given an encrypted polynomial it is possible to perform some polynomial operations, including

1. *Polynomial evaluation:* Given $E_{poly}(P)$ and z it is possible to compute $E(P(z))$. If $P(x) = p_n x^n + \cdots + p_1 x + p_0$, then $E(P(z)) = E(p_n z^n + \cdots + p_1 z + p_0) = E(p_n z^n) * \cdots * E(p_1 z) * E(p_0) = E(p_n)^{z^n} * \cdots * E(p_1)^z * E(p_0)$ which can be computed with knowledge of $E_{poly}(P)$ and z.
2. *Polynomial addition:* Given $E_{poly}(P)$ and $E_{poly}(Q)$ it is possible to compute $E_{poly}(P + Q)$. Assume that $P(x) = p_n x^n + \cdots + p_1 x + p_0$, and $Q(x) = q_n x^n + \cdots + q_1 x + q_0$, if P and Q have different degrees, then one of them can be padded with 0's to make the degrees the same. Now $P + Q(x) = (p_n + q_n)x^n + \cdots + (p_1 + q_1)x + (p_0 + q_0)$, thus $E_{poly}(P + Q)$

Input: Alice has a vector $\vec{A} =< a_1, \ldots, a_n >$ and Bob has a vector $\vec{B} =< b_1, \ldots, b_n >$.
Output: Alice learns $\vec{A} \circ \vec{B}$.

1. *Setup:* Alice creates parameters for a semantically secure additively homomorphic encryption scheme and sends Bob the public parameters; we denote encryption by E. Note that this setup phase has to happen only once for multiple executions of the protocol.
2. Alice sends to Bob the values $E(a_1), \ldots, E(a_n)$.
3. Bob calculates the following value: $E(a_1)^{b_1} * \cdots * E(a_n)^{b_n} * E(0)$. It is straightforward to verify that this value is $E(\vec{A} \circ \vec{B})$, and the multiplication by $E(0)$ is to re-randomize the encryption. Bob sends this value to Alice.
4. Alice decrypts the result and learns $\vec{A} \circ \vec{B}$.

FIGURE 14.5 Two-party HBC protocol for scalar product.

is $E(p_n + q_n), \ldots, E(p_1 + q_1), E(p_0 + q_0)$ which is just $E(p_n) * E(q_n), \ldots, E(p_1) * E(q_1)$, $E(p_0) * E(q_0)$.

3. *Polynomial multiplication:* Given $E_{poly}(P)$ and Q it is possible to compute $E_{poly}(P * Q)$. Assume that $P(x) = p_n x^n + \cdots + p_1 x + p_0$, and $Q(x) = q_m x^m + \cdots + q_1 x + q_0$. Now, $P * Q(x) = s_{n+m} x^{n+m} + \cdots + s_1 x + s_0$ where $s_i = \sum_{j=0}^{i} (p_j * q_{i-j})$.* Furthermore, $E(s_i) = \prod_{j=0}^{i} E(p_j)^{q^{i-j}}$ which can be computed from knowledge of $E_{poly}(P)$ and Q.
4. *Polynomial differentiation:* Given $E_{poly}(P)$, it is possible to compute $E_{poly}(P')$ where P' is the first derivative of P. Note that other derivatives can be computed by repeating this process. Assume that $P(x) = p_n x^n + \cdots + p_1 x + p_0$, then $P'(x) = n p_n x^{n-1} + \cdots + p_1$, and so the new coefficients can be computed with the knowledge of $E_{poly}(P)$.

14.6.4 Set Operations

The ability to perform polynomial operations is useful for computing set operations (set union, set intersection, etc.) [12,13,20]. Set operations are useful for other privacy-preserving computations (such as privacy-preserving data mining). The basic idea behind many of these protocols is to represent the sets as polynomials and then perform set operations on these sets by doing polynomial operations.

To represent a set $S = \{s_1, \ldots, s_m\}$, the polynomial $P_S(x) = (x - s_1) \cdots (x - s_m)$ is used. Note that $P_S(z) = 0$ if and only if $z \in S$.† Using the polynomial operations from the previous section, we can do some types of set operations.

1. *Polynomial evaluation:* Given $E_{poly}(P_S)$, one can compute $E(P_S(z))$, and this will be an encryption of 0 iff $z \in S$. Thus this is a method for detecting if an element is in a set.
2. *Polynomial addition:* The polynomial $P_S + P_T$ will be 0 at all values in $S \cap T$. Thus this is useful for computing set intersection.
3. *Polynomial multiplication:* The polynomial $P_S * P_T$ will be the polynomial that represents the multiset union of S and T.
4. *Polynomial derivation:* The polynomial $P_S{}'$ will be 0 for all items that are in S two or more times, and thus this is useful for eliminating duplicates in a multiset.

The above building blocks can be combined together to form protocols for set intersection and set union. Figure 14.6 is a simplified version of the protocol in [12] for set intersection, and Figure 14.7 describes a simplified protocol for secure two-party set union that was introduced in [13].

14.7 Composing Secure Protocols

One may wonder how to put secure building blocks together to form a secure protocol. However, secure protocols are not always composable. For example, suppose Alice has a point in Cartesian space and that Bob has a point in Cartesian space. Furthermore, suppose that a threshold distance T is known to both Alice and Bob. Furthermore, suppose that the goal of the protocol is for Alice to learn if the distance between her and Bob's points is smaller than T. One way of doing this is to use a secure protocol to compute the distance between the two points and to reveal this value to Alice. Then Alice would compute the result from this value and T. Clearly, the resulting protocol would not be secure, because it reveals the distance between the two points. On a positive note, it

* Note that $p_\ell = 0$ for $\ell > n$ and $q_\ell = 0$ for $\ell > m$.

† It is worth noting that if we are doing modular arithmetic over a large base that the more correct statement is: $P_S(z) = 0$ if $z \in S$ and if $z \notin S$, then $P_S(z) \neq 0$ with high probability.

Input: Alice has a set $A = \{a_1, \ldots, a_n\}$ and Bob has a set $B = \{b_1, \ldots, b_m\}$.
Output: Alice learns $A \cap B$.
Steps:

1. Alice creates a key pair for a semantically secure homomorphic encryption scheme and sends the public parameters to Bob. We denote encryption with this scheme by E and decryption by D.
2. Alice computes a polynomial $P(x) = (x - a_1) * \cdots * (x - a_n) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$. She then sends $E(P) = E(c_n), \ldots, E(c_1), E(c_0)$ to Bob.
3. For each item $b_i \in B$, Bob computes $E(P(b_i) * r_i + b_i)$ for a randomly chosen value r_i and sends all of these values to Alice. Note that $E(P(b_i) * r_i + b_i)$ is $E(b_i)$ if $b_i \in A$ and is a random value otherwise.
4. Alice decrypts all of the values that she receives from Bob and outputs all decrypted values that are in her set.

FIGURE 14.6 Two-party HBC protocol for set intersection.

Input: Alice has a set $A = \{a_1, \ldots, a_n\}$ and Bob has a set $B = \{b_1, \ldots, b_m\}$.
Output: Alice learns $A \cup B$.
Steps:

1. Alice creates a key pair for a semantically secure homomorphic encryption scheme and sends the public parameters to Bob. We denote encryption with this scheme by E and decryption by D.
2. Alice computes a polynomial $P(x) = (x - a_1) * \cdots * (x - a_n) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$. She then sends $E(P) = E(c_n), \ldots, E(c_1), E(c_0)$ to Bob.
3. For each item $b_i \in B$, Bob computes a tuple $(E(P(b_i) * r_i)\ ;\ E(P(b_i) * r_i * b_i))$ for a randomly chosen value r_i and sends all of these values to Alice. Note that if $b_i \in A$ then this tuple will be $(0\ ;\ 0)$ and if $b_i \notin A$ then this tuple will be $(E(R)\ ;\ E(R * b_i))$ for some random value R. In the latter case, b_i is recoverable from the decryption of the tuple by multiplying the second value the inverse of the first value.
4. Alice decrypts all of the tuple that she receives from Bob. If a tuple is $(0\ ;\ 0)$ then she does nothing and otherwise she recovers the value by multiplying the second value the inverse of the first value. She outputs all recovered values along with her own set.

FIGURE 14.7 Two-party HBC protocol for set union.

was shown in [4] that protocols can be composed in certain circumstances. More specifically, if the protocol can be proven secure when each secure building block is replaced with the ideal protocol for the building block (i.e., with the trusted third party solution), then the protocol that uses the secure protocols (instead of the ideal protocols) is also secure.

The following is a contrived example of a secure protocol composition, suppose we are trying to calculate the intersection of two sets A and B that are known respectively to Alice and Bob. Suppose we have two building blocks: (1) CARDINALITY$(A; B)$—which outputs the $|A \cap B|$, and (2) SETINT$(A, |A \cap B|; B)$ that outputs $A \cap B$. In this case one way to compute this value would be to first run CARDINALITY(A, B) to obtain $|A \cap B|$, and then use SETINT$(A, |A \cap B|; B)$ to calculate the result. If the individual protocols SETINT and CARDINALTITY are secure then this composition is also secure, because revealing the cardinality as an intermediate input is not a privacy violation because it is also revealed by the final result.

14.8 Summary

In summary, secure protocols provide a way to compute a function over distributed inputs, without revealing anything other than the result of the protocol. Of course, unless the function is independent of the inputs, the result will reveal some information about the inputs into the protocol, but a secure protocol should not reveal anything about the inputs other than what can be deduced from the output. In some cases, the output may be too revealing, i.e., the output of the function reveals too much

information. However, when the result of a specific information collaboration is not too revealing, secure protocols are a promising technique, because they allow the result of the collaboration to be computed while preserving the privacy of the inputs. In fact, secure protocols have been created for many application domains, including auctions, data mining, set operations, benchmarking, and privacy-preserving surveys.

References

1. M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *CRYPTO '89: Proceedings on Advances in Cryptology*, pages 547–557, Springer-Verlag, New York, 1989.
2. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 1–10, ACM Press, New York, 1988.
3. G. Brassard, C. Crepeau, and J. Robert. All-or-nothing disclosure of secrets. In *CRYPTO*, pages 234–238, Springer-Verlag, London, U.K., 1986.
4. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
5. R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 639–648, ACM Press, New York, 1996.
6. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM symposium on Theory of Computing*, pages 11–19, ACM Press, New York, 1988.
7. C. Crepeau. Equivalence between two flavours of oblivious transfers. In *CRYPTO*, pages 350–354, Springer-Verlag, London, U.K., 1987.
8. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, LNCS 1992, pages 119–136, Springer-Verlag, Berlin, Germany, 2001.
9. I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Advances in Cryptology (CRYPT0 05)*, *LNCS* 3621, pages 378–394, Springer-Verlag, Berlin, Germany, 2005.
10. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.
11. P. Feldman and S. Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 148–161, ACM Press, New York, 1988.
12. M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proceedings of Advances in Cryptology—EUROCRYPT '04*, LNCS, 3027, pages 1–19, Springer-Verlag, Berlin, Germany, 2004.
13. K. Frikken. Privacy-preserving set union. In *ACNS 2007*, LNCS, 2007, pages 237–252, Springer-Verlag, Berlin, Germany, 2007.
14. B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen. On private scalar product computation for privacy-preserving data mining. In *The 7th Annual International Conference on Information Security and Cryptology (ICISC 2004)*, pages 104–120, Seoul, Korea, 2004.
15. O. Goldreich. *Foundations of Cryptography: Volume I Basic Tools*. Cambridge University Press, Cambridge, U.K., 2001.
16. O. Goldreich. *Foundations of Cryptography: Volume II Basic Application*. Cambridge University Press, Cambridge, U.K., 2004.

17. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Conference on Theory of Computing*, pages 218–229, ACM Press, New York, 1987.
18. S. Goldwasser. Multi party computations: Past and present. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–6, ACM Press, New York, 1997.
19. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
20. L. Kissner and D. Song. Privacy-preserving set operations. In *Proceedings of Advances in Cryptology–CRYPTO '05*, LNCS, 3621, 2005. Full version appears at http://www.cs.cmu.edu/leak/.
21. Y. Lindell and B. Pinkas. A proof of yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. http://eprint.iacr.org/.
22. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay—A secure two-party computation system. In *Proceedings of Usenix Security*, pages 287–302, San Diego, CA, 2004.
23. S. Micali and P. Rogaway. Secure computation (abstract). In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 392–404, Springer-Verlag, Berlin, Germany, 1992.
24. P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography Conference (PKC)*, LNCS, 3958, pages 458–473, Springer-Verlag, Berlin, Germany, 2006.
25. M. Naor, B. Pinkas, and Re. Sumner. Privacy preserving auctions and mechanism design. In *EC '99: Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, ACM Press, New York, 1999.
26. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology: EUROCRYPT '99*, LNCS, 1592, pages 223–238, Springer-Verlag, Berlin, Germany, 1999.
27. M. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, Cambridge, MA, 1981.
28. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
29. D. Woodruff. Revisiting the efficiency of malicious two-party computation. In Naor, M. ed., *Eurocrypt 2007*, LNCS, vol. 4515, Springer, Heidelberg, pages 79–96, 2007.
30. A.C. Yao. Protocols for secure computations. In *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pages 160–164, IEEE Computer Society Press, Washington, DC, 1982.
31. A.C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 162–167, IEEE Computer Society Press, Washington, DC, 1986.