Carmit Hazay, Yehuda Lindell

Efficient Secure Two-Party Protocols

Techniques and Constructions

July 14, 2010

Springer

To our families, for all their support

Preface

In the setting of multiparty computation, sets of two or more parties with private inputs wish to jointly compute some (predetermined) function of their inputs. The computation should be such that the outputs received by the parties are correctly distributed, and furthermore, that the privacy of each party's input is preserved as much as possible, even in the presence of adversarial behavior. This encompasses any distributed computing task and includes computations as simple as coin-tossing and broadcast, and as complex as electronic voting, electronic auctions, electronic cash schemes and anonymous transactions. The feasibility (and infeasibility) of multiparty computation has been extensively studied, resulting in a rather comprehensive understanding of what can and cannot be securely computed, and under what assumptions.

The theory of cryptography in general, and secure multiparty computation in particular, is rich and elegant. Indeed, the mere fact that it is possible to actually achieve the aforementioned task is both surprising and intriguing. However, the focus of this book is not on the theory of secure computation (although a number of results with theoretical importance are studied here), but rather on the question of *efficiency*. Recently, there has been increasing interest in the possibility of actually using secure multiparty computation to solve real-world problems. This poses an exciting challenge to the field of cryptography: Can we construct secure protocols (with rigorous proofs of security) that are truly efficient, and thus take the theory of secure computation to the next step towards practice. We stress that this book is not about "practical cryptography". We do not take systems considerations into account, nor how protocols should be implemented and deployed. Instead, our aim is to provide an introduction to the field of efficient protocol construction and design. We hope that this book will make the field of secure computation in general, and efficient protocol construction in particular, more accessible and will increase awareness regarding the importance of this vibrant field.

Outline. This book is divided into three distinct parts:

- **Introduction and definitions:** We begin with a general introduction and survey of secure computation, followed by definitions of security under a number of different adversary models. This part also includes important material regarding the properties of these definitions, and the relations between them.
- **General constructions:** In this part, we present secure protocols for general secure computation. That is, we present protocols that can be applied to any circuit computing any efficient function. Although this does not enable us to utilize specific properties of the function being computed, the resulting protocols can be efficient enough if the circuit and input are not too large.
- **Specific constructions:** Finally, we study secure protocols for specific problems of interest. Two of the chapters in this part consider efficient constructions of basic building blocks that are widely used in constructions of secure protocols; namely, zero-knowledge (via Σ protocols) and oblivious transfer. The last two chapters study two specific examples of higher-level protocols; specifically, the secure computation of the kth ranked element (or median) of a distributed list, and secure search operations on databases. The constructions in this part demonstrate how specific properties of a function being computed can be utilized to achieve greater efficiency.

It goes without saying that the material presented in this book is far from an exhaustive study of results in the field. There are many alternative constructions achieving some of the results presented here, and many other problems of interest for which efficient protocols have been constructed. In some places throughout, we have added pointers to additional readings of relevance.

In order to not unnecessarily complicate the constructions and models, we have focused on the *two-party* case and consider only *static adversaries* and the *stand-alone model*. We do not claim that this is the best model for constructing protocols; indeed it is arguably too weak in many cases. However, we believe that it serves as a good setting for an initial study, as it is significantly cleaner than other more complex settings.

Prerequisite knowledge. We assume that the reader is familiar with the basics of theoretical cryptography. Thus, for example, we assume that readers know what commitment schemes and zero-knowledge proofs are, and that they are comfortable with notions like pseudorandomness and computational indistinguishability. In contrast, all the relevant definitions of secure two-party computation are presented here from scratch. Thus, this book can also be used as a first introduction to secure computation.

Reading this book. Although there are advantages to reading this book in sequential order, much of the book can be read "out of order". It goes without saying that the chapter on definitions is needed for all later chapters. However, it is possible to read definitions as needed (e.g., read Section 2.2

and then Chapter 3, then Section 2.3 followed by Chapter 4, and so on). Regarding the general constructions in Part II of the book, the constructions in Chapters 4 and 5 rely in a direct way on Chapter 3, and thus it is highly recommended to read Chapter 3 first. In contrast, Chapters 4 and 5 can be read independently of each other.

The specific constructions in Part III can be read independently of the general constructions in Part II. It is preferable to read Chapters 6 and 7 first (and in order) because later protocols use the tools introduced in these chapters. In addition, some of the oblivious transfer protocols of Chapter 7 use zero-knowledge proofs that are constructed in Chapter 6. Nevertheless, if one is satisfied with referring to an arbitrary zero-knowledge proof or oblivious transfer protocol, then the chapters in Part III can be read in any order.

Book aims and its use for teaching a course. This book can be used as a textbook for an introductory course on secure computation with a focus on techniques for achieving efficiency, as an entry point for researchers in cryptography and other fields like privacy-preserving data mining who are interested in efficient protocols for secure computation, and as a reference for researchers already in the field. Regarding its use as a textbook, due to the flexibility regarding the order of reading this book (as described above), it is possible to design courses with different focuses. For example, a more theoretical course would spend considerable time on definitions and the general constructions of Part II of the book, whereas a more applied course would focus more on the specific constructions in Part III. We remark also that Chapters 6 and 7 can serve as a nice opening to a course; the material is not as heavy as general secure computation and contains many interesting ideas that can be attractive to students. When teaching a general introduction to (computational) secure computation, it is certainly possible to base much of the course on this book. However, in such a case we would also teach the GMW construction. A full treatment of this appears in [35, Chapter 7].

Comments and errata. We will be more than happy to receive any (positive or negative) feedback that you have on this book, as well as any errors that you may find. Please email us your comments and errata to `lindell@cs.biu.ac.il`. A list of known errata will be maintained at `http://www.cs.biu.ac.il/~lindell/efficient-protocols.html`.

Acknowledgements. First and foremost, we would like to thank Ivan Damgård for generously providing us with the text that formed the basis of Chapter 6 on Σ protocols. In addition, we would like to thank Oded Goldreich, Jonathan Katz and Eran Omri for providing us with constructive advice and comments on this book.

Carmit Hazay: First, I would like to thank my co-author Yehuda Lindell who was also my Ph.D. advisor. Yehuda introduced me to the area of secure computation and has greatly contributed to my academic career. He is a continuing source of inspiration and assistance, and I am grateful to him for an amazing journey which led to this book.

During my Ph.D. I had the pleasure of working with many talented people who enriched my knowledge and deepened my understanding regarding secure computation. I would like to thank Ran Canetti, Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, Kobbi Nissim, Tal Rabin and Hila Zarosim for many productive discussions and a memorable time.

Yehuda Lindell: First and foremost I would like to thank Oded Goldreich. Beyond being my Ph.D. advisor, and as such of great influence on my academic career, Oded has continued to provide valuable support, advice and encouragement. I owe much to Oded and am greatly indebted to him.

The ability to write this book is due to the knowledge that I have gained over many years of research in the field of secure computation. In this time, I have worked with many different co-authors and have benefited from countless fruitful discussions with many members of our research community. I would like to thank Yonatan Aumann, Boaz Barak, Ran Canetti, Rosario Gennaro, Shafi Goldwasser, Shai Halevi, Carmit Hazay, Yuval Ishai, Yael Kalai, Jonathan Katz, Eyal Kushilevitz, Hugo Krawczyk, Tal Malkin, Moni Naor, Benny Pinkas, Tal Rabin, Alon Rosen and Adam Smith for years of joint work and cooperation in a friendly and enjoyable environment. Finally, I would like to give a special thanks to Benny Pinkas for all I have learned from him regarding topics of efficiency in secure protocols.

My work on this project was supported in part by a generous starting grant from the European Research Council.

May 2010 Carmit Hazay and Yehuda Lindell

Contents

Part I
Introduction and Definitions

In the first two chapters of this book we provide a general introduction to the field of secure computation, as well as rigorous definitions for secure two-party computation in multiple models. Specifically, we consider security in the presence of semi-honest and malicious adversaries, as well as introduce the notion of covert adversaries and security that is not based on the full simulation ideal/real-model paradigm.

Chapter 1
Introduction

The focus of this book is on constructing *efficient* secure protocols for the *two-party* setting. In this introduction, we begin with a general high-level survey of secure multiparty computation. This places the topic of this book in its larger context. Following this, we describe the basic results and techniques related to efficiency in secure computation. Finally, we conclude with a roadmap to the book.

1.1 Secure Multiparty Computation – Background

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 that 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 coalition of parties learns anything about the individual votes of other parties, and the correctness requirement ensures that no coalition of parties can influence the outcome of the election beyond just voting for their preferred candidate. Likewise, in an auction protocol, the privacy requirement ensures that only the winning bid is revealed (this may be desired), and the correctness requirement ensures that the highest bidder is indeed the party to win (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, if not every, cryptographic problem (including the classic tasks of encryption and authentication). Therefore, questions of feasibility and infeasibility for secure multiparty computation are fundamental to the theory and practice of cryptography.

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 (where 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, 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 conclude 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 influence 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 just 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. [11] following [37, 5, 59]) 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 over perfectly private channels to the trusted party, which then computes the desired function and passes 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 and so 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 amongst 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 outputs of the adversary and honest parties in a real protocol execution to their outputs in 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 half or more of the participating parties (that is, in the case where 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. In particular, it is impossible for two parties to toss an unbiased coin when one may be corrupt [17]. 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 cause this abort to take place after it has already obtained its output, but before all the honest parties receive 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 and thus all the other properties are still preserved.

Recently it has been shown that in the case of no honest majority, some non-trivial functions can be securely computed with complete fairness [39]. Despite this, we will forgo any attempt at achieving fairness because (a) general constructions cannot achieve fairness due to [17], and (b) we focus on efficient protocols and all currently known techniques for achieving fairness for non-trivial functions are inherently inefficient.

We note that there are works that aim to provide intermediate notions of fairness [77, 29, 6, 37, 40]. However, we limit our reference to the cases that either (complete) fairness and output delivery are guaranteed, or neither fairness (of any type) nor output delivery are guaranteed.

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 just passively gather information or can it instruct the corrupted parties to act maliciously), and what complexity the adversary is assumed to have (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 and 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 whom to corrupt, and when, can be arbitrarily decided by the adversary and may depend on what is has seen throughout 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 [67, 13], considers the possibility that parties are corrupted for a certain period of time only. Thus, honest parties may become corrupted throughout the computation (as in the adaptive adversarial model), but corrupted parties may also become honest.
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, it does model inadvertent leakage of information by honest parties and thus is useful in some cases (e.g., where the parties essentially trust each other but want to ensure that nothing beyond the output is leaked). This model may also be of use in settings where the use of the "correct" software running the correct protocol can be enforced. 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. However, protocols that achieve this level of security are typically much less efficient. Malicious adversaries are also called "active".

   These are the classic adversarial models. However, in some cases, an intermediate adversary model may be required. This is due to the fact that the semi-honest adversary modeling is often too weak, whereas our protocols that achieve security in the presence of malicious adversary may be far too inefficient. An intermediate adversary model is that of covert adversaries. Loosely speaking, such an adversary may behave maliciously. However, it is guaranteed that if it does so, then it will be *caught cheating* by the honest parties with some given probability.
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).
   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 [9, 15] and the *computational* model [77, 35]. 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 on or interfere with the communication between honest parties).

In contrast, in the computational setting the adversary is assumed to be polynomial time. Results in this model typically assume cryptographic assumptions like the existence of trapdoor permutations. 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 [38] and a public-key infrastructure.

It is only possible to achieve information-theoretic security in the case of an honest majority [9]. Thus, it is not relevant to the case of two-party computation, which is the focus of this book. We will therefore consider the computational setting only.

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

Stand-alone computation versus composition. All of the above relates to the stand-alone model, where only a single protocol execution takes place (or many take place but only one is "under attack"). A far more realistic model is that of *concurrent general composition* where many secure (and possibly insecure) protocols are executed together [12]. This is a strictly harder problem to solve [14] and has been the focus of much work in the past decade. See [54] for a study of this topic.

Feasibility of secure multiparty computation. The above-described definition of security seems to be very restrictive in that no adversarial success is tolerated, irrespective of its strategy. 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 for the case of *malicious adversaries* and *static corruptions* in the *stand-alone model*. 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 guaranteed output delivery can be achieved for any function in a point-to-point network, without any setup assumptions. This can be achieved both in the computational set-

ting [35] (assuming the existence of trapdoor permutations) and in the information-theoretic (private channel) setting [9, 15].
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 [35] (under the same assumptions as above), and in the information-theoretic setting [73, 4].
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 for any function assuming that the parties have access to a broadcast channel and in addition assuming the existence of enhanced trapdoor permutations [77, 35, 32]. These feasibility results hold only in the computational setting; analogous results for the information-theoretic setting cannot be obtained when $t \geq m/2$ [9].

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, then fairness and guaranteed output delivery are not obtained. We note that the above results all hold with respect to malicious, static adversaries in the stand-alone model.

This book – two-parties, static adversaries and the stand-alone model. As we have mentioned, adaptive corruption captures a real-world threat and as such protocols that are secure in the presence of such adversaries provide a strong security guarantee. In addition, the stand-alone model of computation is not the realistic model in which protocols are executed today. Nevertheless, the problem of constructing *highly efficient two-party protocols* that are secure in the presence of *static adversaries in the stand-alone model* serves as an important stepping stone for constructing protocols in more complex settings. As we will see, it is already difficult to construct efficient protocols for static adversaries in the stand-alone model, and we strongly believe that a broad understanding of the problems that arise in more restricted settings is needed before progressing to more complex settings. Our experience also shows us that the techniques developed for solving the problems of secure computation in the stand-alone model with static adversaries are often useful also in the more complex setting of concurrent composition (with static or adaptive adversaries). For these reasons, we have chosen to focus solely on the stand-alone model and static adversaries in this book.

1.2 The GMW Protocol for Secure Computation

As we have mentioned above, it has been shown that any probabilistic polynomial-time two-party functionality can be securely computed in the presence of malicious adversaries (without fairness or guaranteed output delivery), assuming the existence of enhanced trapdoor permutations [35, 32]. This powerful feasibility result – known as the GMW construction – is obtained in two stages. First, it is shown how to securely compute any functionality in the presence of semi-honest adversaries. Then, a protocol compiler is presented that takes any protocol that is secure in the presence of semi-honest adversaries and outputs a protocol that is secure in the presence of malicious adversaries.

Security for semi-honest adversaries. A secure protocol is constructed based on a Boolean circuit that computes the functionality in question. The basic idea behind the construction is for the parties to iteratively compute the gates in the circuit in an "oblivious manner". This is achieved by having the parties first share their input bits; that is, for every input wire to the circuit, the parties hold random bits α and β so that $\alpha \oplus \beta$ equals the actual input bit associated with that wire. Then, for every (AND/OR/NOT) gate, the parties run a mini-protocol to compute random shares of the output of the gate, based on their given random shares of the inputs to the gate. At the end of the protocol, the parties hold random shares of the output wires which they can send to each other in order to reconstruct the actual output. The security of the protocol is derived from the fact that each party sees only random values (shares) throughout the protocol. Therefore, it learns nothing beyond the output, as required.

Compilation to security for malicious adversaries. The basic idea that stands behind the GMW construction is to have the parties run a suitable protocol that is secure in the presence of semi-honest adversaries, while *forcing* the potentially malicious participants to behave in a semi-honest manner. The GMW compiler therefore takes for input a protocol that is secure against semi-honest adversaries; from here on we refer to this as the "basic protocol". Recall that this protocol is secure in the case where each party follows the protocol specification exactly, using its input and uniformly chosen random tape. We must therefore force a malicious adversary to behave in this way. First and foremost, this involves forcing the parties to follow the prescribed protocol. However, this only makes sense relative to a *given* input and random tape. Furthermore, a malicious party must be forced into using a *uniformly chosen* random tape. This is because the security of the basic protocol may depend on the fact that the party has no freedom in setting its own randomness.[1]

[1] A good example of this is the semi-honest 1-out-of-2 oblivious transfer protocol of [25]. The oblivious transfer functionality is defined by $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$. In the protocol

In light of the above discussion, the GMW protocol compiler begins by having each party commit to its input. Next, the parties run a coin-tossing protocol in order to fix their random tapes (clearly, this protocol must be secure against malicious adversaries). A regular coin-tossing protocol in which both parties receive the same uniformly distributed string is not sufficient here. This is because the parties' random tapes must remain secret. This is solved by augmenting the coin-tossing protocol so that one party receives a uniformly distributed string (to be used as its random tape) and the other party receives a commitment to that string. Now, following these two steps, each party holds its own uniformly distributed random tape and a commitment to the other party's input and random tape. Therefore, each party can be "forced" into working consistently with the committed input and random tape.

We now describe how this behavior is enforced. A protocol specification is a deterministic function of a party's view consisting of its input, random tape and messages received so far. As we have seen, each party holds a commitment to the input and random tape of the other party. Furthermore, the messages sent so far are public. Therefore, the assertion that a new message is computed according to the protocol is of the $\mathcal{NP}$ type (and the party sending the message knows an adequate $\mathcal{NP}$-witness to it). Thus, the parties can use zero-knowledge proofs to show that their steps are indeed according to the protocol specification. As the proofs used are zero-knowledge, they reveal nothing. Furthermore, due to the soundness of the proofs, even a malicious adversary cannot deviate from the protocol specification without being detected. We thus obtain a reduction of the security in the malicious case to the given security of the basic protocol against semi-honest adversaries.

Efficiency and the GMW construction. The complexity of the GMW protocol for semi-honest adversaries is related to the size of the circuit needed to compute the functionality. Specifically, the parties need to run an oblivious transfer (involving asymmetric computations) for every circuit of the gate. Although this results in a significant computational overhead, it is reasonable for functionalities with circuits that are not too large. We remark that Yao's protocol for secure two-party computation is typically more efficient than the GMW construction. This is due to the fact that only symmetric operations are needed for computing every gate of the circuit, and oblivious transfers are only used for the input bits. This means that Yao's protocol scales better

of [25], the receiver gives the sender two images of an enhanced trapdoor permutation, where the receiver knows only one of the preimages. The protocol works so that the receiver obtains x_i if it knows the ith preimage (and otherwise it learns nothing of the value of x_i). Thus, were the receiver to know both preimages, it would learn both x_0 and x_1, in contradiction to the security of the protocol. Now, if the receiver can "alter" its random tape, then it can influence the choice of the images of the permutation so that it knows both preimages. Thus, the fact that the receiver uses a truly random tape is crucial to the security.

to large circuits than GMW. In Chapter 3 we present Yao's protocol for semi-honest adversaries in detail.

As we have mentioned, the cost of achieving security in the presence of semi-honest adversaries is not insignificant. However, it is orders of magnitude less than the cost of achieving security in the presence of malicious adversaries. This is due to the fact that general zero-knowledge protocols for $\mathcal{NP}$ require a Karp reduction from a complex computational statement to a language like 3-colorability or Hamiltonicity. Thus, even when the underlying protocol for semi-honest adversaries is highly efficient, the compiled protocol for malicious adversaries is typically not. We conclude that the GMW construction, and in particular the compilation of GMW from security in the presence of semi-honest adversaries to security in the presence of malicious adversaries, is to be viewed as a fundamental feasibility result, and not as a methodology for obtaining protocols in practice.[2]

Despite what we have stated above, the GMW compilation paradigm *has* had considerable influence over the construction of *efficient protocols*. Indeed, one way to efficiently achieve security in the presence of malicious adversaries is to design a protocol that is secure in the presence of semi-honest adversaries (or a different notion that is weaker than security in the presence of malicious adversaries) in a particular way so that one can efficiently prove "correct behavior" in zero-knowledge. One example of a protocol that uses this paradigm can be found in Section 7.4.

In conclusion, the GMW construction proves that any efficient functionality can be securely computed, even in the presence of a powerful malicious adversary. The next step, given this feasibility result, is to construct more efficient protocols for this task with the final aim of obtaining protocols that can be used in practice. This research goal is the focus of this book.

1.3 A Roadmap to the Book

This book is divided into three distinct parts. We now describe in detail the contents of each part and the chapters therein.

1.3.1 Part I – Introduction and Definitions

In this chapter, we have provided a brief overview of the basic notions, concepts and results of secure computation. The aim of this overview is to place

[2] We stress that this should in no way be interpreted as a criticism of GMW; the GMW construction is a beautiful proof of the feasibility of achieving secure computation and is one of most fundamental results of theoretical cryptography.

the material covered in this book in its general context. In Chapter 2 we present a number of different definitions of secure two-party computation. We begin by presenting the classic definitions of security in the presence of semi-honest and malicious adversaries. As we have discussed above, on the one hand, the security guarantee provided when considering semi-honest adversaries is often insufficient. On the other hand, although protocols that are secure in the presence of malicious adversaries provide a very strong security guarantee, they are often highly inefficient. This motivates the search for alternative definitions that provide satisfactory security guarantees, and that are more amenable to constructing highly efficient protocols. We consider three such relaxations:

1. *Covert adversaries (Section 2.4):* Physical security in the real world is achieved via deterrence. It is well known that an expert thief can break into almost anybody's house and can steal most cars. If this is the case, then why aren't there more expert thieves and why are most of our houses and cars safe? The answer to this is simply deterrence: most people do not want to go to jail and so choose professions that are within the law. (Of course, there are also many people who do not steal because it is immoral, but this is not relevant to our discussion here.) The notion of security in the presence of covert adversaries utilizes the concept of deterrence in secure computation. Specifically, a protocol that achieves security under this notion does not provide a foolproof guarantee that an adversary cannot cheat. Rather, it guarantees that if an adversary does attempt to cheat, then the honest parties will detect this with some given probability (say 0.5 or 0.9). Now, if such a protocol is run in a context where cheating can be penalized, then this level of security can suffice. For example, if a secure protocol is used by a consortium of cellphone companies who wish to carry out a statistical analysis of the usage behaviors of cellphone users, then a cheating company (who tries to steal customer data from its competitors) will be penalized by removing them from the consortium.

   We remark that security in the presence of malicious adversaries is the analogue of an armed security guard outside your house 24 hours a day. It is much safer to protect your house in this way. However, the costs involved are often not worth the gain.
2. *Non-simulation based definitions (Section 2.6):* The definitions of security for semi-honest, malicious and covert adversaries all follow the ideal/real-model simulation-based paradigm. We consider two relaxations that do not follow this paradigm.

   a. *Privacy only (Section 2.6.1):* As we have discussed, the simulation-based method of defining security (via the ideal/real-model paradigm) guarantees privacy, correctness, independence of inputs and more. However, in some cases, it may suffice to guarantee privacy without the other properties. For example, if a user wishes to search a database so that her search queries are kept private, then privacy alone may suffice.

b. *One-sided simulation (Section 2.6.2):* In many cases, it is very difficult to formalize a definition of security that guarantees privacy only. This is due to the fact that when a party receives output it learns something and we must try to state that it should learn nothing more. However, the output depends on the parties' inputs and if these are not explicit then it is unclear what the output should be. In contrast, it is very easy to define privacy when nothing should be learned; in such a case, privacy can be formalized via indistinguishability in the same way as encryption. The notion of one-sided simulation helps to define security for protocol problems in which only one party is supposed to receive output. In such a case, we require simulation (via the ideal/real-model paradigm) for the party that receives input, and privacy only (via indistinguishability) for the party that does not receive output. Observe that correctness and independence of inputs are not guaranteed when the party who does not receive output is corrupted. However, as in the example for privacy only above, this is sometimes sufficient. The advantage of "one-sided simulation" over "privacy only" is that a general definition can be given for any functionality in which only one party receives output.

We stress that the "right definition" depends very much on the application being considered. In some cases, it is crucial that security in the presence of malicious adversaries be achieved; take for example computation over highly confidential data that can cause significant damage if revealed. However, in many other cases, weaker notions of security can suffice, especially if the alternative is to not use a secure protocol at all (e.g., as may be the case if the best protocols known for a task that provide security for malicious adversaries are not efficient enough for use).

In addition to presenting the above definitions of security, Chapter 2 contains the following additional material. In Section 2.3.3 we discuss the surprising fact that due to a quirk in the definitions, security in the presence of malicious adversaries does not always imply security in the presence of semi-honest adversaries. In Section 2.5 we show that in many cases it suffices to consider restricted types of functionalities, enabling a simpler presentation. Finally, in Section 2.7 we state modular sequential composition theorems that are very useful when proving the security of protocols.

1.3.2 Part II – General Constructions

A general construction is a protocol that can be used for securely computing any functionality. These constructions are typically based on a circuit for computing the functionality, and as such do not utilize any special properties of the functionality being computed. Thus, they cannot be used for complex computations applied to very large inputs. Despite this, it is important to

study these constructions for the following reasons. First, many useful techniques and methodologies can be learned from them. Second, in many cases, a larger protocol uses a smaller subprotocol that is obtained via a general construction (an example of this is given in Chapter 8). Finally, as the efficiency of general constructions improves, we are able to use them for more and more real problems [22, 71].

Semi-honest adversaries. In Chapter 3 we present Yao's protocol for achieving secure two-party computation in the presence of semi-honest adversaries [77]. This protocol works by having one party prepare an encrypted or garbled version of the circuit that can be decrypted to yield only one value, the output of the computation. When the circuit being computed is not too large, this protocol is very efficient. Specifically, the parties need $O(1)$ asymmetric computations per input bit, and $O(1)$ symmetric computations per gate of the circuit. In practice, symmetric computations are far more efficient than asymmetric computations. Thus, a circuit with hundreds of thousands of gates can be easily computed.

Malicious adversaries. In Chapter 4 we present a protocol that achieves security in the presence of malicious adversaries [55]. This protocol is based on Yao's protocol for the semi-honest case, and includes significant machinery for preventing the parties from cheating. The basic technique for achieving this is called cut-and-choose. Specifically, one of the main problems that arises when running Yao's protocol with malicious adversaries is that the party who constructs the garbled circuit can construct it incorrectly (since it is encrypted, this cannot be detected). In order to prevent such behavior, we have the party construct many copies of the circuit and then ask it to open half of them. In this way, we can be sure that most of the remaining unopened circuits are correct. It turns out that this intuitive idea is very hard to implement correctly, and many new problems arise when computing with many circuits. As a result, the construction is much less efficient than in the semi-honest case. However, it can still be run on circuits with tens of thousands of gates, as will be discussed below.

Covert adversaries. In Chapter 5 we present a protocol that is based on the same idea as that in Chapter 4 but provides security only in the presence of covert adversaries. The main idea is that in the context of covert adversaries it suffices to use cut-and-choose on many fewer circuits, and it suffices to compute only one circuit at the end. This results in a protocol that is much more efficient than that required to achieve security in the presence of malicious adversaries. Roughly speaking, when the adversary is guaranteed to be caught with probability ϵ if it attempts to cheat, the cost of the protocol is about $O(1/\epsilon)$ times the cost of Yao's semi-honest protocol. Thus, for $\epsilon = 1/2$ it is possible to compute circuits that contain hundreds of thousands of gates, as in the semi-honest case.

Implementations of general protocols. Recent interest in the field of efficient protocols has led to implementations that are useful for understanding the real efficiency behavior of the above protocols. One work which is of relevance here is an implementation of a protocol for securely computing the AES function [71]. That is, one party holds a secret 128-bit symmetric key k for the AES function and the other party holds a 128-bit input x. The computation is such that the first party learns nothing about x, while the second party learns $AES_k(x)$ and nothing else. Such a functionality has many applications, as we will see in Chapter 9. In [71], the exact protocols of Chapters 3, 4 and 5 were implemented for a circuit computing AES which has approximately 33,000 gates. The protocols were implemented using a number of different optimizations. The best optimizations yielded running times of seven seconds for the semi-honest protocol, 95 seconds for the covert protocol and 1,148 seconds for the malicious protocol. Although these running times are not fast enough for real-time applications, they demonstrate that it is feasible to carry out such computations on circuits that are large (tens of thousands of gates). We expect that further efficiency improvements will not be long coming, and believe that these times will be significantly reduced in the not too distant future (especially for the malicious case).

1.3.3 Part III – Specific Constructions

As we have mentioned, the drawback of considering general constructions is that it is not possible to utilize special properties of the functionality being computed. In the final part of the book, we present protocols for specific problems of interest. This part is also divided into two subparts. First, in Chapters 6 and 7 we present some basic tools that are very useful for designing efficient protocols. Then, in Chapters 8 and 9 we study two specific problems as a demonstration of how higher-level protocols can be constructed.

Sigma protocols and efficient zero-knowledge. In Chapter 6 we show how highly efficient zero-knowledge protocols can be constructed. As we have discussed, security in the presence of malicious adversaries is typically achieved by forcing the parties to behave honestly. The immediate way to do this is to force the parties to prove in zero-knowledge that they are following the protocol specification. Needless to say, a straightforward implementation of this idea is very inefficient. For this reason, many try to stay clear of explicit zero-knowledge proofs for enforcing honest behavior. However, in many cases it *is* possible to construct a protocol for which the zero-knowledge proof that is needed is highly efficient. Many of these efficient zero-knowledge protocols are constructed from a simpler primitive called a Σ-protocol. In Chapter 6 we study Σ-protocols in depth and, among other things, present highly efficient generic transformations from Σ-protocols to zero-knowledge proofs of membership and zero-knowledge proofs of knowledge. These transformations

are very useful because it is far easier to construct a protocol and prove that it is a Σ-protocol than to construct a protocol and prove that it is a zero-knowledge proof of knowledge.

Oblivious transfer and applications. In Chapter 7 we construct oblivious transfer protocols that are secure under the definitions of privacy only, one-sided simulation, and full simulation-based security in the presence of malicious adversaries. The protocols that are presented progress in a natural way from privacy only through one-sided simulation to full security. The final protocols obtained have only a constant number of exponentiations and as such are very efficient. In addition, we present optimizations for the case where many oblivious transfers need to be run, which is the case in many secure protocols using oblivious transfer. We then conclude with protocols for pseudorandom function evaluation, which is a primitive that also has many applications.

The *k*th-ranked element and search problems. In Chapters 8 and 9 we show how to securely compute the kth-ranked element of two lists (with a special case being the median) and how to search databases and documents in a secure manner. Admittedly, the choice of these two problems is arbitrary and is based on our personal preferences. Nevertheless, we believe that they are interesting examples of how specific properties of the functionality in question can be used to solve the problem with high efficiency.