Diss. ETH No. 17102

Approaches to Efficient and Robust Cryptographic Protocols

A dissertation submitted to

ETH ZURICH

for the degree of
Doctor of Sciences

presented by

Bartosz Jan Przydatek
Dipl. Informatik-Ing. ETH

born on August 23, 1972, in Poland
citizen of Poland

accepted on the recommendation of

Prof. Dr. Ueli Maurer, examiner
Prof. Dr. Ronald Cramer, co-examiner
Dr. Martin Hirt, co-examiner

2007

Acknowledgments

I would like to thank my advisor, Ueli Maurer, not only for conveying to me a passion for cryptography, but also for his guidance, patience, and generosity. I am very grateful to Ronald Cramer and Martin Hirt for agreeing to be co-referees of this thesis. The results presented here stem from collaborations with Martin, Remo Meier, Jesper Nielsen, and Jürg Wullschleger, and I would like to thank them for sharing their thoughts and ideas with me. Furthermore, I would like to thank Danny Harnik and Omer Reingold for encouraging comments on parts of this work.

During my graduate studies at CMU and at ETH I have greatly enjoyed the pleasant atmosphere, stimulating discussions and continued support also from many other collaborators, colleagues and friends, including Yevgeniy Dodis, Stefan Dziembowski, Matthias Fitzi, Abie Flaxman, Anya Goldenberg, Michelle Grant, Thomas Holenstein, Yan Karklin, Robert König, Anton Likhodedov, Aria Llitjós, Rob Miller, Adrian Perrig, Krzysztof Pietrzak, Renato Renner, Peter Richter, Sanjit Seshia, Johan Sjödin, Dawn Song, Reto Strobl, Stefano Tessaro, Douglas Wikström, Stefan Wolf, Ke Yang, Oskar Zięta, Martin Zinkevich, and many others. Special thanks go to Peter, Sanjit, and Rob, with whom I have shared my office at CMU, and to Krzysztof and Johan, my office-mates at ETH. Moreover, I owe many thanks to Manuel Blum, Ernst Specker, and Emo Welzl, from whom I have learned a lot about science and about life.

I would like also to thank Christian Cachin, Jan Camenisch, and Rafi Ostrovsky, for their hospitality and great mentoring during my internships at IBM Research and at DIMACS.

I am deeply grateful to my parents, to my sister, and to other members of my family for their encouragement and unconditional support.

Finally, and above all, I would like to thank Drzach — for everything!

Zurich, in May 2007

Abstract

The growing influence of the Internet and other communication networks on our daily lives and on the global economy shows clearly that the security issues in such networks are of the uttermost importance. Motivated both by the theoretical questions and by the potential real-world applications, in this dissertation we study problems of secure cooperation in communication networks. In particular we focus on constructing efficient and robust protocols for various cryptographic tasks.

In the first part of this thesis we study the problem of secure *multi-party computation* (MPC), which allows a set of n players to evaluate an agreed function of their inputs in a secure way, i.e., so that an adversary corrupting some of the players cannot achieve more than controlling the inputs and outputs of these players. The concept of MPC is very general and powerful, since it allows to realize essentially *any* distributed computational task in a *secure* way. For that reason the MPC problem has been studied extensively since its introduction by Yao in 1982. A major goal of these studies is to design protocols with low communication complexity, and two main research directions emerged over the time, with focus on reducing round-, resp. bit-complexity. In this thesis we focus on the bit-complexity, i.e., the number of bits communicated between the parties during the computation, and we consider this problem in *asynchronous* networks, which model pretty closely real-world networks. We propose an MPC protocol, which is secure with respect to an active adversary corrupting up to $t < n/3$ players (this is optimal in an asynchronous network), and which is the most efficient protocol currently known. For our constructions we develop several novel techniques, which were used also in subsequent works on efficient MPC protocols.

In the second part of this thesis we turn to a problem which is common to all cryptographic research based on computational assumptions.

In spite of considerable advances in theoretical computer science and specifically in complexity theory, it is still not known whether there exist provably hard problems that could form a solid foundation for the complexity-based cryptography. In particular, it is not clear which assumptions are the best, and which are the most likely to hold in 10 or 20 years from now. Therefore, when implementing a cryptographic system in a real-world setting we are confronted with the difficult problem of choosing the most trustworthy assumptions. One way of dealing with this problem is offered by the so-called *robust combiners*, i.e., constructions which in a certain sense allow to build cryptographic systems based on the *best assumption possible*, without actually being able to tell which assumption is the best one. Roughly speaking, a robust combiner *combines* several implementations of a primitive based on various assumptions, to yield an implementation guaranteed to be secure if at least *some* of the underlying assumptions (i.e. sufficiently many but not necessarily all) are valid. We generalize the notion of robust combiners in several ways, and propose constructions of combiners for various fundamental primitives, like private information retrieval (PIR), oblivious transfer (OT) and oblivious linear function evaluation (OLFE). Our constructions offer trade-offs between applicability and efficiency, and are strictly stronger and/or more efficient than the constructions known before. Moreover, we introduce *cross-primitive* combiners, which can be viewed as a generalization of reductions and combiners. Using this more general view we show a separation between PIR and OT, ruling out certain types of reductions of PIR to OT.

Zusammenfassung

Die wachsende Rolle des Internets und anderer Kommunikationsnetzwerke im täglichen Leben und in der globalen Wirtschaft demonstriert deutlich, dass die Sicherheitsgarantien in solchen Netzen von äusserst grosser Bedeutung sind. Motiviert durch die theoretischen Fragestellungen und durch die potentielle Anwendungen werden in dieser Dissertation verschiedene Probleme der Kooperation und des verteilten Rechnens in den Kommunikationsnetzen studiert. Insbesondere konzentrieren wir uns auf das Konstruieren von effizienten und robusten Protokollen für ausgewählte kryptographische Aufgaben.

Im ersten Teil dieser Arbeit studieren wir das Problem vom sicheren *Multi-Party Computation* (MPC), das einer Menge von n Spielern erlaubt eine beliebige, vereinbarte Funktion von ihren Inputs auf eine sichere Weise zu berechnen. Dabei werden insbesondere die Geheimhaltung und die Korrektheit garantiert: die Inputs der einzelnen Spieler bleiben geheim, und selbst ein Gegner, der einige der Spieler korrumpiert, nicht mehr erreichen kann als die Inputs und die Outputs der korrumpierten Spieler zu manipulieren. Das Konzept von MPC ist sehr generell und umfangreich, da es erlaubt im Wesentlichen jede verteilte Berechnung auf sichere Weise zu realisieren. Aus diesem Grund wurde das MPC-Problem seit dessen Formulierung durch Yao im Jahr 1982 sehr intensiv studiert. Ein Hauptziel von vielen Arbeiten auf diesem Gebiet besteht darin, Protokolle mit niedriger Kommunikationskomplexität zu entwerfen, wobei sich über die Zeit zwei Forschungsrichtungen etabliert haben, die den Entwurf von Protokollen mit kleiner Runden- bzw. Bit-Komplexität anstreben. In dieser Dissertation konzentrieren wir uns auf die Bit-Komplexität, d.h. die Anzahl Bits, die zwischen den Spielern während der Berechnung kommuniziert werden. Dabei betrachten wir das MPC-Problem in einem *asynchronen* Netzwerk, das ein realitätsnahes Modell von heutigen Kommunikationsnetzwerken darstellt. Wir entwerfen ein MPC-Protokoll, das zur Zeit das effizienteste bekannte Protokoll ist. Das Protokoll garantiert Sicherheit gegen einen aktiven Gegner, der bis zu den $t < n/3$ Spielern korrumpiert, was in einem asynchronen Netzwerk optimal ist. Für unsere Konstruktionen entwickeln wir einige neue Techniken, welche auch in den späteren Arbeiten über effiziente MPC-Protokolle verwendet werden.

Im zweiten Teil dieser Dissertation betrachten wir ein Problem, das alle kryptographische Systeme betrifft, deren Sicherheit auf berechnungsmässigen Annahmen basiert. Trotz der beträchtlichen Fortschritte in der theoretischen Informatik und insbesondere in der Komplexitätstheorie ist nach wie vor nicht bekannt, ob es beweisbar schwierige Berechnungsprobleme gibt, welche eine feste Grundlage für die komplexitätsbasierte Kryptographie bilden könnten. Insbesondere ist es nicht klar, welche berechenmässigen Annahmen am besten sind, und welche in etwa 10 oder 20 Jahren noch korrekt sein werden. Wenn man also ein kryptographisches System implementieren möchte, wird man mit einer schwierigen Frage konfrontiert: Welche Annahme ist am vertrauenwürdigsten? Eine Lösung dieses Dilemmas sind sog. *robuste Combiner*, d.h. Konstruktionen von kryptographischen Systemen, deren Sicherheit im gewissen Sinne auf *den besten* Annahmen basiert ist, ohne dass dabei entschieden werden muss, welche Annahmen eigentlich am besten sind. Ein robuster Combiner nimmt als Input mehrere Implementierungen einer Primitive, die auf verschiedenen Annahmen basieren, und konstruiert eine Implementierung, die garantiert sicher ist, wenn mindestens einige (d.h. genug viele, aber nicht notwendigerweise alle) der Annahmen gültig sind. Wir verallgemeinern und erweitern den Begriff von robusten Combiners, und schlagen Combiner-Konstruktionen für verschiedene grundlegende Primitive vor, wie z.B. *Private Information Retrieval* (PIR), *Oblivious Transfer* (OT) und *Oblivious Linear Function Evaluation*. Unsere Konstruktionen bieten *Tradeoffs* zwischen Anwendbarkeit und Effizienz an, und sind wesentlich stärker und/oder effizienter als die bereits bekanten Konstruktionen. Ferner führen wir die sog. *cross-primitive Combiners* ein, die als eine Verallgemeinerung von robusten combiners und von Reduktionen gesehen werden können. Mit dieser Verallgemeinerung zeigen wir eine Separation zwischen PIR und OT, welche gewisse Typen von Reduktionen von PIR zu OT ausschliesst.

Contents

Chapter 1

Introduction

Traditionally cryptography was concerned mainly with confidentiality and authenticity of information, ensured by the encryption and the authentication of messages. While these basic tasks still belong to the core of cryptography, in the 20th century this area has evolved into a rich discipline on the intersection of computer science and mathematics, and is closely related to system security, computational complexity, number theory and information theory. In particular, the past 30 years, since the seminal paper of Diffie and Hellman [DH76], have been particularly fruitful in cryptography, and have given a rise to a number of beautiful, and often paradoxical, concepts and their realizations, like public-key cryptography [DH76, Mer78, RSA78], zero-knowledge proofs [GMR89], or multiparty computation [Yao82, GMW87, BGW88, CCD88]. These fundamental results combined with the rapid development and the growing popularity of the Internet and other communication networks, have helped to establish cryptography as an active research area, attracting both practitioners and theoreticians.

On the practical side, it is fair to say that we are just at the beginning of the information age, and our dependence on digital communication and media will only grow. Therefore it is crucial to ensure the reliability and the security of the digital world, especially since new possibilities and opportunities come together with new dangers and risks. Of course, for secure solutions to be relevant in practice, it is essential that they are *efficient* in terms of the computation and the communication overhead.

On the other hand, there are also numerous theoretical impulses that fuel the cryptographic research. Many new concepts and primitives give a rise to the questions about relationships between them, which in turn call for a deeper study and understanding of the proposed ideas. Moreover, as in many areas of theoretical computer science, also in cryptography the mere existence of a solution or feasibility in principle is often neither sufficient nor satisfactory, and the ultimate goal is to find solutions that are as efficient as possible, and/or to prove the optimality of existing solutions. In this sense showing the principal existence of a solution to a problem is just the beginning rather than the end, and constitutes a major driving force for further research on the problem.

Motivated both by the theoretical questions and by the potential real-world applications, in this thesis we study problems of secure cooperation in communication networks. In particular we focus on constructing efficient and robust protocols for various cryptographic tasks.

In the first part of this thesis we study the problem of secure *multi-party computation* (MPC), which allows a group of mutually mistrusting players to compute any function of their private inputs, without disclosing the inputs, while the privacy of the inputs and the correctness of the outputs should be guaranteed even if some of the players cheat and do not follow the protocol. Motivated by both theoretical and practical considerations, we focus on the design of efficient solutions for this problem. Even though the fundamental feasibility results for MPC were presented almost 20 years ago [Yao82, GMW87, BGW88, CCD88], the area is far from being closed. While the problem of finding the most efficient solutions for various models is definitely important from the theoretical point of view, it is also of interest from the practical side. In particular, if provably secure and *efficient* solutions are available, they are more likely to be implemented in real-world networks (instead of ad-hoc, seemingly secure solutions).

In the second part we focus on dangers of a different kind. In spite of considerable achievements in theoretical computer science, and in particular in complexity theory, it is still not known whether there exist provably hard problems that could form a solid foundation for the complexity-based cryptography. In other words, modern cryptographic tools and protocols, on which the bulk of today's secure communication and e-commerce is based, rely on various unproven assumptions which potentially could be invalidated any time, for example through a discovery of some ingenious algorithm (cf. recent attacks on Secure Hash Algorithm (SHA) [WYY05b, WYY05a]). While there exist a few intensively studied and relatively well understood assumptions, it is definitely not clear which assumptions are the best, and in particular which are the most likely to hold in 10 or 20 years from now. Moreover, new concepts and primitives come often together with new assumptions, allowing efficient implementations but also introducing new risks. Therefore, when implementing a cryptographic system in a real-life setting we are often confronted with the problem of choosing the most trustworthy assumptions.

Possible ways of dealing with this problem include taking a guess (maybe even an educated one), or using only information-theoretic cryptography, which is not based on any assumptions. While the first way is probably the most common in practice, it clearly fails if we have bad luck. The second way is definitely secure, but it misses many of the most exciting cryptographic tools and concepts, which are possible only with the complexity-based cryptography.

In this thesis we pursue a different way, which in a certain sense allows to construct cryptographic systems based on the best assumption possible, without actually being able to tell which assumption is the best one. Such constructions, so-called *robust combiners*, combine several implementations of a primitive based on various assumptions, and yield an implementation guaranteed to be secure if at least *some* assumptions (i.e. sufficiently many but not necessarily all) are valid.

In the rest of this chapter we describe the studied problems in more detail, and give an overview of the contributions of this thesis.

1.1 Secure multi-party computation

The goal of secure multi-party computation (MPC) is to allow a set of n players to evaluate an agreed function of their inputs in a secure way, where "secure" means that an adversary corrupting some of the players cannot achieve more than controlling the inputs and outputs of these players. In particular, the adversary does not learn the inputs of the uncorrupted players, and furthermore, she cannot influence the outputs of the uncorrupted players except by selecting the inputs of the corrupted players.

The problem of secure multi-party computation has been extensively studied in a variety of models differing in many aspects, like for example the underlying communication model (synchronous vs. asynchronous, with vs. without broadcast channel), or the limitations of the adversary (computationally bounded vs. unbounded, static vs. adaptive, threshold vs. general).

In this work we focus on *asynchronous* communication, i.e., the messages in the network can be delayed for an arbitrary amount of time (but eventually all messages are delivered). As a worst-case assumption, we give the ability of controlling the delay of messages to the adversary. The asynchronous communication models many real-world networks, like the Internet, much better than the synchronous communication. However, it turns out that MPC protocols for asynchronous networks are significantly more involved than their synchronous counterparts. One reason for this is that in an asynchronous network, when a player does not receive an expected message, he cannot distinguish whether the sender is corrupted and did not send the message, or the message was sent but delayed in the network. This implies also that in a fully asynchronous setting it is impossible to consider the inputs of *all* uncorrupted players when evaluating the function. The inputs of some (potentially honest) players have to be ignored, because waiting for them could turn out to be endless [Bec54].

History and related work The MPC problem was first proposed by Yao [Yao82] and solved by Goldreich, Micali, and Wigderson [GMW87] for computationally bounded adversaries, and by Ben-Or, Goldwasser, and Wigderson [BGW88], and independently by Chaum, Crépeau, and Damgård [CCD88] for adversaries that are computationally unbounded. All these protocols considered a synchronous network with a global clock. The first MPC protocol for the asynchronous model (with unconditional security) was proposed by Ben-Or, Canetti, and Goldreich [BCG93]. Extensions and improvements, still in the unconditional model, were proposed in [BKR94, SR00, PSR02]. A great overview of asynchronous MPC with unconditional security is given in [Can95].

Following the fundamental results on principal feasibility of MPC, there has been a vast body of work on the *efficiency* of MPC protocols, thus trying to enable their applicability *in practice*. As mentioned above, usually the main efficiency bottleneck of distributed systems is the need of interaction and communication between the participants, while the local computations are relatively simple and inexpensive. Thus a major goal in these efforts is to design protocols with low communication complexity. In particular, a number of works focused on designing MPC protocols requiring only few rounds of communication (e.g. [BB89, BMR90, BFKR90, CDI05]), or minimizing the bit complexity, i.e. the number of bits communicated during the computation (e.g. [BFKR90, FY92, GRR98, CDD^{+}99, HMP00, SR00, CDN01, CDF01, HM01, PSR02, HN06]).

The most efficient asynchronous protocols known previously are the protocols of Srinathan and Rangan [SR00] and of Prabhu, Srinathan and Rangan [PSR02]. The first one requires $\Omega(n^2)$ invocations to the broadcast primitive for every multiplication, which makes the protocol very inefficient when broadcast is realized with some asynchronous broadcast protocol. The latter protocol is rather efficient, as it requires $\Omega(n^4\kappa)$ bits of communication per multiplication. However, it tolerates only $t < n/4$ corruptions, which is not optimal.

Other directions of research on MPC focus on the primitives and algebraic structures sufficient for MPC. In particular, a line of work investigated the possibility of basing secure computation on secret sharing (cf. [CDD00, CDM00, CDG$^+$05]). More recently, Chen and Cramer [CC06] showed how to base MPC on special types of codes, rather than polynomials, which results in MPC based on fields of constant size, rather then fiels of size linear in the number of players (see also [CCG$^+$07]).

1.2 Robust combiners

Informally speaking, a *robust combiner* is a construction which protects against wrong cryptographic implementations or assumptions. Consider a scenario when two candidate implementations, C_1 and C_2, of some cryptographic primitive are given, e.g., two encryption schemes or two bit commitment schemes. Each implementation is based on some unproven computational assumption, a_1 resp. a_2, like for example the hardness of factoring integer numbers or the hardness of computing discrete logarithms. We would like to have an implementation C of the primitive, which is as secure as possible given the current state of knowledge. As it is often not clear, which of the assumptions a_1, a_2 is more likely to be correct, picking just one of the implementations does not work — we might bet on the wrong assumption! A better option would be to have an implementation which is guaranteed to be secure as long as *at least one* of the assumptions a_1, a_2 is correct. That is, given C_1 and C_2 we would like to construct an efficient implementation C, which is secure whenever at least one of the input implementations is. Such a construction is an example of a *(1;2)-robust combiner*, as it *combines* the input implementations and is *robust* against situations when one of the two inputs is insecure.

In general, robust combiners can use more than two input schemes, and aim at providing a secure implementation of the output primitive assuming that sufficiently many of the candidates are secure. More formally, for $\mathcal{A}$ denoting a cryptographic primitive, like for example a one-way function, a *$(k; m)$-robust $\mathcal{A}$-combiner* is a construction which takes m implementations of $\mathcal{A}$ as input and yields an implementation of $\mathcal{A}$, which is guaranteed to be secure as long as at least k input implementations are secure.

The concept of robust combiners is actually not so new in cryptography and many techniques are known for combining cryptographic primitives to improve security guarantees, e.g., cascading of block ciphers. However, a more formal and rigorous study of combiners was initiated quite recently [Her05, HKN$^+$05].

Robust combiners for some cryptographic primitives, like one-way functions or pseudorandom generators, are rather simple, while for others, e.g., for oblivious transfer (OT), the construction of combiners seems to be considerably harder. In particular, in a recent work Harnik *et al.* [HKN$^+$05] show that there exists no "transparent black-box" $(1; 2)$-robust OT-combiner. This is rather unfortunate, since oblivious transfer is a fundamental primitive on which many other cryptographic applications can be based [Kil88].

In this thesis we explore in more depth the problem of combining oblivious transfer, and also study the combiners of other cryptographic primitives. Given the impossibility result for OT-combiners, it is particularly interesting to investigate the existence of combiners for (single-database) *private information retrieval* (PIR), a primitive closely related, yet not known to be equivalent, to oblivious transfer. Potential PIR-combiners could lead to better understanding of relations between PIR, OT, and other primitives. Moreover, constructions of PIR-combiners are also of considerable practical interest, stemming from the fact that some of the most efficient PIR protocols are based on relatively new computational assumptions (e.g., [CMS99, KY01]), which are less studied and thus potentially more likely to be proved wrong.

History and related work As mentioned above, a more rigorous study of robust combiners was initiated only recently, by Herzberg [Her05] and by Harnik *et al.* [HKN$^+$05]. In particular, Boneh and Boyen [BB06], and Pietrzak [Pie07] studied the efficiency of combiners for collision resistant hash functions. On the other hand, there are numerous implicit uses and constructions of combiners in the literature (e.g., [AB81, EG85, MM93, DK05, HL05]).

Private information retrieval was introduced by Chor *et al.* [CKGS98] and has been intensively studied since then. The original setting of PIR consists of multiple non-communicating copies of the database and guarantees *information-theoretic* privacy for the user. Later, Kushilevitz and Ostrovsky [KO97] gave the first solution to the *single-database* PIR, in which the privacy of the user is based on a computational assumption. The first PIR protocol with communication polylogarithmic in the size of the database was proposed by Cachin *et al.* [CMS99], and recently more efficient constructions have been proposed (e.g. [Cha04, Lip05]). Further information about PIR can be found in a survey by Gasarch [Gas04].

The relationships between PIR and other primitives have been studied intensively in the recent years. In particular, Beimel *et al.* [BIKM99] proved that any non-trivial single-database PIR implies one-way functions, and Di Crescenzo *et al.* [DMO00] showed that such a PIR implies oblivious transfer. Kushilevitz and Ostrovsky [KO00] demonstrated that one-way trapdoor *permutations* are sufficient for non-trivial single-database PIR. On the negative side, Fischlin [Fis02] showed that there is no black-box construction of one-round (i.e., two-message) PIR from one-to-one trapdoor *functions*.

Techniques similar to the ones employed in the proposed robust PIR-combiners were previously used by Di Crescenzo *et al.* [DIO01] in constructions of universal service-providers for PIR.

1.3 Contributions of this thesis

The scope and the contributions of this thesis can be divided in two parts. In the first part we study the problem of secure multi-party computation in asynchronous networks, and focus on the bit complexity. We propose protocols for MPC, which are significantly more efficient than the solutions known so far, and achieve optimal resilience. In the second part we construct robust combiners for various fundamental cryptographic primitives, and we use the insights from these constructions to study the relationship between the primitives. Short summaries of contributions from each part are described in the subsections below.

1.3.1 Asynchronous multi-party computation

We study the problem of MPC in asynchronous networks and propose a number of new protocols for this problem. First we present a basic protocol, following the paradigm of MPC based on a threshold homomorphic encryption scheme, as introduced by Franklin and Haber [FH96] and made robust by Cramer, Damgård and Nielsen [CDN01]. We adapt this paradigm to asynchronous networks, to obtain a conceptually simple and relatively efficient MPC protocol. Then we propose a number of transformations and improvements, which lead to the most efficient protocol known to date.

The proposed protocols are for the cryptographic model, in which it is assumed, that the adversary is computationally bounded, and they are secure with respect to an active adversary corrupting up to $t < n/3$ players, which is optimal in an asynchronous network [Tou84, BT85].

As mentioned above, our focus is on achieving low bit complexity. Once the inputs are distributed, the final proposed protocol with a security parameter κ needs only $\mathcal{O}(c_M n^2 \kappa)$ bits of communication to evaluate a circuit with c_M multiplication gates. This is only by a factor $\mathcal{O}(n)$ worse than the bit complexity of the most efficient protocol known for *synchronous* networks, due to Hirt and Nielsen [HN06]. Moreover, the proposed protocol improves on the communication complexity of the most efficient optimally-robust asynchronous MPC protocol, due to Srinathan and Pandu Rangan [SR00], by a factor of $\Omega(n^2)$. In contrast to the protocol of [SR00], our protocol uses broadcast only in a very limited manner: the number of broadcast invocations is independent of the size of the circuit. This nice property is also achieved in [PSR02], but this protocol is non-optimal (it tolerates only $t < n/4$) and requires $\Omega(n^2)$ times more communication than ours.

In the asynchronous MPC, the agreed function can be evaluated only on a subset of the inputs, i.e., some (potentially honest) player cannot provide their input into the computation. However, the presented protocol can easily be extended to consider the input of each (honest) party, at the cost of few rounds of synchronous communication during the input stage. We believe that this *hybrid* model is quite a realistic one, and mimics the real world even more closely than the fully asynchronous model. After all, even though the networks like the Internet are asynchronous, synchronization succeeds in daily lives, e.g. by using also other means of communication like telephones.

In our constructions we employ some novel techniques, which can be of independent interest. In fact, some of the techniques proposed in this work were recently used to construct the most efficient MPC protocol for *synchronous* networks, with linear communication cost per multiplication gate [HN06].

The results on asynchronous MPC presented in this thesis are a joint work with Martin Hirt and Jesper Buus Nielsen [HNP05, HNP06].

1.3.2 Robust combiners of cryptographic primitives

The results of our study of robust combiners fall into two main categories, definitional and constructional. The first category contains new definitions of combiners, which are more general and/or stronger than the definitions used previously. The second category encompasses numerous constructions of robust combiners of specific cryptographic primitives, for both the old definitions and the newly proposed ones.

The primitives which we consider in the context of robust combiners include private information retrieval (PIR), oblivious transfer (OT), and bit commitment (BC). In particular, we propose a $(1;2)$-robust PIR-combiner, i.e. combiner which given two implementations of PIR yield an implementation of PIR which is secure if at least one of the input implementations is secure. We also describe various techniques and optimizations based on properties of existing PIR protocols, which yield PIR-combiners with better efficiency and applicability.

Furthermore, we introduce a notion of "cross-primitive" combiners, which can be viewed as a generalization of combiners and reductions. More formally, for $\mathcal{A}$ and $\mathcal{B}$ denoting cryptographic primitives, a *$(k;m)$-robust $\mathcal{A}$-to-$\mathcal{B}$ combiner* is a construction which takes m implementations of $\mathcal{A}$ as input and yields an implementation of $\mathcal{B}$, which is guaranteed to be secure as long as at least k input implementations are secure. We present $(1;2)$-robust PIR-to-BC and PIR-to-OT combiners, which are the first proposed cross-primitive combiners. While interesting in their own right, such combiners also offer insights into relationships and reductions between cryptographic primitives. In particular, our PIR-to-OT combiner together with the impossibility result of [HKN+05] rule out certain types of reductions of PIR to OT.

Moreover, we take a closer look at the notion of combiners and suggest a more fine-grained approach to the design of such constructions. That is, we argue that in order to obtain combiners as efficient as possible, the constructions may take into account that some properties of the input candidates are proved to hold unconditionally, and hence cannot fail even if some computational assumption turns out to be wrong. Therefore, keeping in mind the original motivation for combiners, i.e. protection against wrong assumptions, a more fine-grained approach to design of robust combiners exploits the unconditionally secure properties and focuses on the properties which hold only under given assumptions.

This change in the approach yields sometimes immediately trivial constructions of combiners (as observed by Harnik *et al.* [HKN$^+$05] for OT and BC), yet in many cases the resulting problems are still interesting and challenging. Motivated by these observations, we propose a new, stronger, and more general definition of robust combiners for two-party primitives. The new definition captures scenarios where in the candidate implementations the security of Alice is based on an assumption different from the assumption underlying Bob's security, or where the security of one party is unconditional. This finer distinction can then be exploited in constructions of combiners.

For this new definition we propose OT-combiners yielding secure OT when the total number of candidates' failures on side of either party is strictly smaller than the number of candidates. In particular, we propose an OT-combiner which guarantees secure OT even when only one candidate is secure for both parties and each of the remaining candidates is insecure for one of the parties. Furthermore, we present a combiner for *oblivious linear function evaluation* (OLFE), a primitive which can be viewed as a generalization of oblivious transfer. The proposed construction is optimal both with respect to the robustness and in terms of the use of the input candidates. Finally, we propose also efficient *uniform* combiners for OT and OLFE, i.e. constructions which are secure *simultaneously* for a wide range of candidates' failures.

We show the optimal robustness of the proposed combiners by proving a *very simple, yet stronger* impossibility result for OT-combiners. While the impossibility proof in [HKN$^+$05] only proves the non-existence of *transparent black-box* combiners, our proof excludes *all types* of combiners. Moreover, since our definition is stronger than the previous definition, all constructions satisfy also the latter, and we obtain tight bounds also for the previous definition.

The results on robust combiners presented in this thesis are a joint work with Remo Meier and Jürg Wullschleger [MP06, MPW07, PW06].

Chapter 2

Asynchronous multi-party computation

In this chapter we consider the problem of secure *multi-party computation* (MPC) in asynchronous networks. We present a number of protocols for this problem, achieving an optimal resilience against an active adversary and a very low communication complexity. Since we focus on *asynchronous* networks, the proposed protocols can actually be directly implemented in real-world networks like the Internet.

In the presented protocols we follow the paradigm of MPC based on a threshold homomorphic encryption scheme, as introduced by Franklin and Haber [FH96], who presented a protocol secure in the *honest-but-curious* model. Later Cramer, Damgård and Nielsen [CDN01] proposed an efficient protocol resilient also against an active adversary. However, this protocol uses synchronous communication in an essential manner.

This chapter is organized as follows. After presenting the model and describing the tools we use in our constructions, we show how to remove the dependence on synchronous communication present in the previously proposed protocols, by introducing a technique to relax the requirements of player cooperation — every player leads the computation on his own, and requests help of other players only when necessary. This results in an *asynchronous* protocol, which however is less efficient than the synchronous counterparts. In the subsequent sections we propose improvements, which allow to reduce the communication overhead, yielding the currently most efficient MPC protocol with optimal resilience. Finally, we discuss various extensions to the model and the protocols, which lead to solutions applicable to many real-world situations. The results presented in this chapter are a joint work with Martin Hirt and Jesper Buus Nielsen [HNP05, HNP06].

2.1 Formal model and preliminaries

Notation. Throughout this chapter we use n to denote the number of players (i.e., parties) participating in the MPC protocol, we use $P_1, \ldots, P_n$ to denote the players, and we use $\mathcal{P}$ to denote the set of all players. For an integer $m > 0$ we write $[m]$ to denote the set $\{1, \ldots, m\}$. Our constructions are parametrized by a security parameter κ.

2.1.1 Communication model

Motivated by the current real-world communication networks, like the Internet, we consider a model in which the communication takes place over an *asynchronous*, public network, which enables the participating parties access to point-to-point authenticated channels, but without guaranteed delivery of messages.

More formally, an n-player protocol is a tuple $\pi = (P_1, \ldots, P_n, \mathsf{init})$, where each P_i is a probabilistic interactive Turing machine, and init is an *initialization function*, used for the usual set-up tasks, like initialization of the players, setting up cryptographic keys and public parameters, etc. The players communicate over an asynchronous network, in which the delay between sending and delivery of a message is unbounded. More precisely, when a player sends a message, this message is added to the set of messages already sent but not yet delivered, $\mathsf{Msg} = \{(i, j, m)\}$, where (i, j, m) denotes a message m from P_i to P_j. The delivery of the messages is scheduled by the adversary (see below).

2.1.2 The general MPC model

We use the model of security of asynchronous protocols proposed by Canetti [Can00]. Formally our model for running a protocol will be the hybrid model with a functionality for distributing some initial cryptographic keys between the parties using some function init. The ideal functionality that we wish to realize is given by a circuit Circ, which consists of input gates augmented by the party to supply the input, linear and multiplication gates defining the actual computation, and output gates augmented by the party to see the output. The circuit Circ is defined over some ring $\mathcal{M}$, which is a public parameter resulting from the initialization function init.

Adversary. We consider an adversary which is bounded by a polynomial in the security parameter κ. The adversary controls the delivery of all messages and can corrupt up to t parties. A corrupted party is under full control of the adversary, which sees all incoming messages, and determines all outgoing messages. The adversary schedules the delivery of the messages arbitrarily, by picking a message $(i, j, m) \in \mathsf{Msg}$ and delivering it to the recipient. The adversary doesn't see the contents of messages exchanged between honest (i.e., not corrupted) parties, and any message from an honest party to an honest party is *eventually* delivered. In most cases we require that $t < n/3$, but will sometimes consider other thresholds. The set of parties to be corrupted is specified by the adversary *before* the execution of the protocol, i.e., we consider static security.

Execution of a protocol. Before the protocol starts, an initialization function init is evaluated on random input $r \in \{0,1\}^*$ to generate a tuple $(\mathrm{sv}_1, \ldots, \mathrm{sv}_n, \mathrm{pv}) = \mathsf{init}(1^\kappa, r)$ of secret values sv_i and a public value pv, where 1^κ is an unary encoding of the security parameter. Each party P_i is initialized with $(1^\kappa, \mathrm{sv}_i, \mathrm{pv})$. At the beginning of the protocol execution, every party P_i receives its input value x_i from the environment, and produces some initial messages $(i, \cdot, \cdot)$ which are added to the set Msg. The adversary is given the public value pv, the values (x_j, sv_j) for each corrupted party P_j, and the control over the set Msg. Subsequently the protocol is executed in a sequence of *activations*. In each activation the adversary picks a message $(i, j, m) \in \mathsf{Msg}$ and delivers it to P_j. Upon delivery of a message, party P_j performs some computation based on its current state, updates its state and produces some messages of the form $(j, \cdot, \cdot)$, which are added to the set Msg. In some activation the parties can produce the output to the environment and terminate. The adversary determines the inputs x_i and all messages of corrupted parties. The adversary and the environment can communicate with each other.

Security. The security of a protocol is defined relative to an ideal evaluation of the circuit by requiring that for any adversary attacking the execution of the protocol there exists a simulator which can simulate the attack of the adversary to any environment given only an ideal process for evaluating the circuit. In the ideal process the simulator has very restricted capabilities: It sees the inputs of the corrupted parties. Then it specifies a subset $W \subseteq [n]$ of the parties to be the input providers, under the restriction that $|W| \geq n - t$. The set W determines which parties provide inputs to the computation of a circuit Circ, and the adversary specifies the input values of the corrupted parties. The input gates of Circ belonging to the parties from W are assigned the inputs of the corresponding parties, and the remaining input gates are assigned default input values. Then Circ is evaluated and the outputs of the corrupted parties are shown to the simulator.

Given these capabilities the simulator must then simulate to the environment the entire view of an execution of the protocol, including the messages sent and the possible communication between the environment and the adversary.

2.1.3 Efficiency measures

As mentioned in Chapter 1, usually the main efficiency bottleneck of distributed systems is the need of communication between the parties. Therefore, our goal is to design protocols with low communication complexity, while ensuring also their computational efficiency. More precisely, we focus on *bit* complexity, i.e. the number of bits communicated between the parties during the computation. The round complexity[1] of the proposed solutions is linear in the depth of the circuit Circ.

2.2 Cryptographic primitives and protocols

In the proposed MPC protocols we employ a number of standard primitives and sub-protocols. We first introduce the required notation and tools with their essential properties, and then point to the literature to example implementations. We stress that all the needed implementations can be realized based on standard cryptographic assumptions, without use of the random-oracle methodology [BR93, CGH98, MRH04].

[1] Intuitively, the round complexity of a protocol in an asynchronous network is defined as the maximal round complexity of a protocol execution, where the round complexity of an execution is equal to the total time of the execution as measured by an imaginary external clock, divided by the longest delay of a message in the execution [CR93].

2.2.1 Homomorphic encryption with threshold decryption

We assume the existence of a semantically secure probabilistic public-key encryption scheme, which additionally is homomorphic and enables threshold decryption, as specified below.

Encryption and decryption. For an encryption key e and a decryption key d, let $\mathscr{E}_e : \mathcal{M} \times \mathcal{R} \to \mathcal{C}$ denote the encryption function mapping a plaintext $x \in \mathcal{M}$ and a randomness $r \in \mathcal{R}$ to a ciphertext $X \in \mathcal{C}$, and let $\mathscr{D}_d : \mathcal{C} \to \mathcal{M}$ denote the corresponding decryption function. We require that $\mathcal{M}$ is a ring $\mathbb{Z}_M$ for some $M > 1$, and we use "$\cdot$" to denote multiplication in $\mathcal{M}$. We often use capital letters to denote encryptions of the plaintexts denoted by the corresponding lower-case letters. When keys are understood, we write $\mathscr{E}$ and $\mathscr{D}$ instead of $\mathscr{E}_e$ resp. $\mathscr{D}_d$, and we frequently omit the explicit mention of the randomness in the encryption function $\mathscr{E}$.

Homomorphic property. We require that there exist (efficiently computable) binary operations $+$, $*$, and $\oplus$, such that $(\mathcal{M}, +)$, $(\mathcal{R}, *)$, and $(\mathcal{C}, \oplus)$ are algebraic groups, and that $\mathscr{E}_e$ is a group homomorphism, i.e.

$$\mathscr{E}(a, r_a) \oplus \mathscr{E}(b, r_b) = \mathscr{E}(a + b, r_a * r_b) \ .$$

We use $A \ominus B$ to denote $A \oplus (-B)$, where $-B$ denotes the inverse of B in the group $\mathcal{C}$. For an integer a and $B \in \mathcal{C}$ we use $a \star B$ to denote the sum of B with itself a times in $\mathcal{C}$.

Ciphertext re-randomization. For $X \in \mathcal{C}$ and $r \in \mathcal{R}$ we let $\mathscr{R}_e(X, r) = X \oplus \mathscr{E}_e(0, r)$. We use $X' = \mathscr{R}_e(X)$ to denote $X' = \mathscr{R}_e(X, r)$ for a uniformly random $r \in \mathcal{R}$. We call $X' = \mathscr{R}_e(X)$ a re-randomization of X. Note that X' is a uniformly random encryption of $\mathscr{D}_d(X)$.

Threshold decryption. We require that there exists a threshold function sharing of decryption $\mathscr{D}_d$ among n parties. More precisely, we assume that for a construction threshold $t_{\mathsf{D}} = t + 1$, there exists a sharing $(d_1, \ldots, d_n)$ of the decryption key d (where d_i is intended for party P_i), satisfying the following conditions. Given the decryption shares $x_i = \mathscr{D}_{i,d_i}(X)$ for t_{D} distinct decryption-key shares d_i, it is possible to efficiently compute x such that $x = \mathscr{D}_d(X)$. Furthermore, the encryption scheme should be still semantically secure against chosen plaintext attack when the adversary is given $t_{\mathsf{D}} - 1$ decryption-key shares. Finally, we require that given a ciphertext X, plaintext $x = \mathscr{D}_d(X)$, and a set of $t_{\mathsf{D}} - 1$ decryption-key shares $\{d_i\}$, it is possible to compute *all* n decryption shares $x_j = \mathscr{D}_{j,d_j}(X)$, $j = 1, \ldots, n$. When keys are understood, we write $\mathscr{D}_i(X)$ to denote the function computing decryption share of party P_i for ciphertext X, and $x = \mathscr{D}(X, \{x_i\}_{i\in I})$ to denote the process of combining the decryption shares $\{x_i\}_{i\in I}$ to a plaintext x.

Robustness. To efficiently protect against cheating servers we require that there exists an efficient two-party zero-knowledge protocol for proving the correctness of a decryption share. For this purpose each share d_i of the decryption key d comes with a related public *verification key* v_i. The required zero-knowledge proof of correctness of a decryption share $x_i = \mathscr{D}_{i,d_i}(X)$ takes (e, v_i, X, x_i) as instance, and (i, d_i) and randomness r as witness. We require also that there exists an efficient two-party zero-knowledge protocol for proving the knowledge of a plaintext, given (e, X) as instance and the corresponding plaintext x and randomness r as witness. Finally, we require an efficient two-party zero-knowledge "proof of correct multiplication" (cf. [CDN01]): given a triple of ciphertexts (R, U, X), with r, u, x denoting the corresponding plaintexts, one shows that U is a re-randomized encryption of the product $u = r \cdot x$. We require that all these protocols communicate $\mathcal{O}(\kappa)$ bits per proof.

2.2.2 Digital signatures

We assume the existence of a digital signature scheme unforgeable against an adaptive chosen message attack. For a signing key s and a verification key v, let $\mathsf{Sign}_s : \{0,1\}^* \to \{0,1\}^\kappa$ denote the signing function, and let $\mathsf{Ver}_v : \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}$ denote the verification function, where $\mathsf{Ver}_v(x, \sigma) = 1$ indicates that σ is a valid signature on the message x. We write $\mathsf{Sign}_i/\mathsf{Ver}_i$ to denote the signing/verification operation of party P_i.

2.2.3 Threshold signatures

We assume the existence of a *threshold* signature scheme, which is unforgeable against an adaptive chosen message attack. For a signing key s and a verification key v, let $\mathscr{S}_s : \{0,1\}^* \to \{0,1\}^\kappa$ denote the signing function, and let $\mathscr{V}_v : \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}$ denote the verification function, where $\mathscr{V}_v(m, \sigma) = 1$ indicates that σ is a valid signature on m.

Threshold signing. We require that there exists a threshold function sharing of $\mathscr{S}_s$ among n parties. That is, we assume that for a given signing threshold t_S, $1 < t_\mathsf{S} \leq n$, there exists a sharing $(s_1, \ldots, s_n)$ of the signing key s (where s_i is intended for party P_i), such that given signature shares $\sigma_i = \mathscr{S}_{i,s_i}(x)$ for t_S distinct signing-key shares s_i, it is possible to efficiently compute a valid signature σ, satisfying $\mathscr{V}_v(x, \sigma) = 1$. We will always have $t_\mathsf{S} = n - t$. The threshold signature scheme should be still unforgeable against adaptive chosen message attack when the adversary is given $(t_\mathsf{S} - 1)$ signing-key shares. Finally, we require that given a signature σ on x, and $(t_\mathsf{S} - 1)$ signing-key shares $\{s_i\}_{i \in I}$, it is possible to compute *all* n signature shares $\sigma_j = \mathscr{S}_{j,s_j}(x)$, $j = 1, \ldots, n$. When keys are understood, we use $\mathscr{S}_i(x)$ to denote the function computing a signature share of party P_i for the message x, and $\sigma = \mathscr{S}(x, \{\sigma_i\}_{i \in I})$ to denote the process of combining the signature shares $\{\sigma_i\}_{i \in I}$ to a signature σ.

Robustness. To protect against cheating servers we require that there exists an efficient two-party zero-knowledge protocol for proving the correctness of a signature share $\sigma_i = \mathscr{S}_{i,s_i}(m)$, given (v, m, σ_i) as instance and given (i, s_i) as witness. We require that this protocol communicates $\mathcal{O}(\kappa)$ bits per proof.

2.2.4 Byzantine Agreement

We require the existence of a Byzantine Agreement (BA) protocol, i.e. a protocol in which the input of every party P_i is a bit $v_i \in \{0,1\}$, the output of every P_i is a bit $w_i \in \{0,1\}$, and which satisfies the following properties. If all honest parties enter the BA, then the BA eventually terminates (*termination*). Upon termination the outputs of all honest players are equal, i.e. $w_i = w$ for some $w \in \{0,1\}$ (*consistency*; we say then that BA terminated with output w). If the BA terminates with output w, then some honest party P_i entered the BA with input $v_i = w$ (*validity*). In particular, if all honest parties have the same input $v_i = v$, then the output of the BA is $w = v$.

2.2.5 Cryptographic assumptions & instantiations of tools

The security of our constructions is based on *decisional composite residuosity assumption* (DCRA) [Pai99]. Alternatively, it could be based also on QRA and strong RSA. We stress, that our constructions are in the plain model. In particular, our constructions *do not* make use of random oracles [BR93, CGH98, MRH04].

Homomorphic encryption with threshold decryption. An example of an encryption scheme satisfying all required properties is Paillier's cryptosystem [Pai99] enhanced by threshold decryption as in [FPS00, DJ01]. In this scheme $\mathcal{M} = \mathbb{Z}_N$ for an RSA modulus N. Another example can be based on the QR assumption and the strong RSA assumption [CDN01].

Digital signatures. As our digital signature scheme we use RSA-based signatures [RSA78], like for example *Probabilistic Signature Scheme* (PSS) by Bellare and Rogaway [BR96].

Threshold signatures. As an example we can use the threshold signature scheme by Shoup [Sho00]. The security of the threshold signature scheme in [Sho00] is based on the assumption that standard RSA signatures are secure. As presented in [Sho00] the zero-knowledge proofs are non-interactive but for the random-oracle model. The protocol can be modified to be secure in the plain (random oracle devoid) model by using interactive proofs (cf. [Nie02]).

Byzantine Agreement. In our protocols we employ the efficient Byzantine Agreement protocol of Cachin *et al.* [CKS00], which has expected constant round complexity, and expected bit complexity of $\mathcal{O}(n^2\kappa)$. As presented in [CKS00] the security proof of the protocol needs the random-oracle methodology (for the above mentioned threshold signatures). This protocol also can be modified to be secure in the plain model [Nie02].

2.3 Kings & slaves: cryptographic MPC with optimal resilience

Our protocols follow the paradigm of multi-party computation based on threshold homomorphic encryption scheme [FH96, CDN01]. In such protocols an encryption key of a public-key encryption scheme is publicly known, while the corresponding decryption key is shared among all the players, so that only the authorized subsets of the players can decrypt any ciphertext (by cooperating in a decryption protocol).

Given such a setup the evaluation of a circuit proceeds as follows. First the parties provide their inputs as ciphertexts of the encryption scheme. Then they cooperate to evaluate the circuit gate-by-gate: given encryptions of inputs of a gate, parties compute an encryption of the corresponding output of the gate, while maintaining privacy of the intermediate values (due to the homomorphic property of the encryption in use, linear gates can be evaluated locally, without any cooperation or communication). Finally, after an encryption of the output gate is computed, parties decrypt this encryption to learn the output.

To achieve robustness against corrupted parties one could use general zero-knowledge proofs, with which each party would prove the correctness of its actions. However, the resulting protocols would be too inefficient to be used *in practice*. Cramer *et. al.* [CDN01] employed *special-purpose* zero-knowledge proofs to obtain an efficient protocol for synchronous communication model. Below we describe how to implement this paradigm in an asynchronous network, achieving simultaneously communication efficiency and optimal resilience. Formally, the main result of this section is summarized in the following theorem.

Theorem 1 *Assuming the existence of homomorphic public-key encryption and digital signatures (cf. Sections 2.2.1 and 2.2.2), there exists a protocol allowing n parties connected by an asynchronous network to securely evaluate any circuit, even in the presence of a computationally bounded adversary actively corrupting up to $t < n/3$ parties.*

The bit complexity of the protocol is $\mathcal{O}(c_I n^3 \kappa + (c_M + c_O) n^4 \kappa)$, where c_I, c_M, c_O denote the number of input, multiplication, and output gates, respectively, and κ is a security parameter. The round complexity of the protocol is linear in the depth of the circuit.

2.3.1 A high-level overview.

The protocol proceeds in three stages, an *input stage*, an *evaluation stage*, and a *termination stage*. In the following, we briefly summarize the goal of each stage.

- *Input stage*: Every player provides an encryption of his input to every other player, and the players jointly agree on a subset of players who have correctly provided their inputs.
- *Evaluation stage*: Every player independently evaluates the circuit on a gate-by-gate basis, with help of the other players. The circuit consists of linear gates, multiplication gates, and output gates.
- *Termination stage*: In parallel to the evaluation stage every player runs also the termination stage, in which it is ensured that every player eventually receives the output(s), and hence every player who is still in the evaluation stage can safely abort it.

By having *every* player evaluate the circuit on his own, we bypass the inherent problems of the asynchronous model. We denote the player that evaluates the circuit as the *king*, and all other players as *slaves* (who support the king). Note that every player acts in parallel once as a king, and n times as a slave, once for every king. In particular, each player acts also as a slave for his own king. The kings are not synchronized among each other; it can happen that one king has almost completed the evaluation of the circuit, while another king is still at the very beginning. However, each slave is synchronized with his king. As soon as the first king completes the evaluation and provably reveals all outputs, all other kings (and their slaves) can safely truncate their own evaluation.

Strictly speaking, the presented protocol is limited to the evaluation of *deterministic* circuits only. Suitable modifications extending the approach to *randomized* circuits are pretty straightforward and are described shortly afterwards, in Section 2.6.

2.3.2 Certificates

In order to achieve robustness we must require every party to prove (in zero-knowledge) the correctness of essentially every value she provides during the protocol execution. To implement this process efficiently we introduce *certificates*, which are used for certifying the truth of claims. Any party can verify the correctness of a certificate locally, without any interaction. Moreover, a certificate should provide no other information than the truth of the claim. Finally, since a certificate is just a bit-string, a party can convince any other party about the truth of the corresponding claim by sending the certificate to the other party, i.e. a certificate can be used to transfer conviction about the truth of a claim.

Below we describe the certificates in more detail, but we intentionally omit a fully formal treatment of this notion. We use certificates as a shorthand and a presentation tool, to allow for an intuitive, yet precise and succinct descriptions of the protocols. Later, in the analysis and security proofs, we consider in each case the corresponding particular implementation of the certificates, which reintroduces the necessary level of formalism.

We say that a bit-string α *is a certificate for claim* m if there exits a publicly known, efficiently computable verification procedure V, such that the following conditions are satisfied, except with negligible probability: if $V(\alpha, m) = 1$ then claim m is true (soundness), and α gives no other information than the truth of the claim m (zero-knowledge). Moreover we require (limited[2]) completeness, i.e. the ability of generating certificates for true claims needed in our protocols. Typical claims used in the proposed protocols include:

(i) «P_i *knows the plaintext of* X_i»

(ii) «*the plaintext of* X_i *is from the set* $\{0, 1, 2\}$»

(iii) «*at least* $n - t$ *parties have received* X_i»

(iv) «X_i *is the unique input of* P_i»

If X is some value, and α is a certificate for some claim m about X (e.g., claim (ii) above), then we say that X *is a value certified (by* α*) for claim* m. Since a certificate is just a bit-string, it is transferable: if a party has constructed or obtained a certificate, she can convince any other party non-interactively of the validity of the claim, by just forwarding the certificate to the other party, who can then locally verify its validity.

In addition to the above conditions, in many cases we require that the certificates for correctness/validity of some data X imply also *uniqueness* of the data, i.e. that it is not possible to obtain two valid certificates for two different values for the same claim. This can be easily achieved by assigning in advance unique identifiers to every gate, every wire and every step in the protocols, and requiring that the identifiers are parts of the claims, e.g. «X_i *is input of* P_i *for wire* id», and that parties participate in construction of at most one certificate for a particular claim. The identifiers can depend on the targeted application of the data, or on party P_i generating it. Occasionally, to clarify the issues, we explicitly specify

[2]The completeness is limited, since we don't require ability of creating certificates for *every* true claim, but just for the claims we need in our constructions.

the use of identifiers, and we emphasize the dependence of id on some parameter(s) $x, y, \ldots$ by writing $\mathsf{id}(x, y, \ldots)$. However, for simplicity we often omit the details from the description of the protocols.

Constructing certificates. For many claims (e.g. (i) and (ii) above) the corresponding certificates could easily be constructed in the random-oracle model [BR93] (by using Fiat-Shamir heuristics [FS86]), but this would be at the costs of a non-standard model. However, there are other useful claims (e.g. some "global" claims like (iii) and (iv) above), which are not directly provable using 2-party interactive proofs, and so obtaining a certificate for them, even in the random-oracle model, is not so straightforward. We therefore follow another approach, which both is realizable in the standard model and allows us to construct certificates for any claim needed in our protocols.

A simple implementation of certificates can be obtained from any signature scheme $(\mathsf{Sign}, \mathsf{Ver})$: a certificate α for claim m is just a set of at least $n-t$ correct signatures: $\alpha := \{\sigma_i\}_{i \in I}$, where $|I| \geq n-t$ and each σ_i is a signature of party P_i on message m, i.e., $\sigma_i = \mathsf{Sign}_i(m)$. The verification procedure V of such a certificate checks all the signatures:

$$V(\{\sigma_i\}_{i \in I}, m) = \begin{cases} 1 & \text{if } |I| \geq n-t \text{ and } \forall i \in I : \mathsf{Ver}_i(m, \sigma_i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Depending on the context and the claim to be certified, we'll use two methods of generating certificates:

bilateral proofs: when a party P_i needs a certificate for knowledge of some secret value, or for validity of some NP-statement (cf. examples (i) and (ii) above, respectively), we will use two-party zero-knowledge proofs. First P_i bilaterally proves the claim m in zero-knowledge to every party P_j, who then, upon successful completion of the proof, sends to party P_i a signature $\sigma_j = \mathsf{Sign}_j(m)$. Party P_i constructs then a certificate α_i as a set of $n-t$ such valid signatures stemming from different parties. In this case we say that "*P_i constructs certificate α_i for «some claim» by bilateral, zero-knowledge proofs*", denoted as

$$\alpha_i := \mathsf{certify}_{\mathsf{zkp}}(\text{«}\textit{some claim}\text{»}) \; .$$

protocol-driven: For other claims, like the "global" examples (iii) and (iv) above, P_i will also construct a certificate α_i as a set of $n-t$ individual signatures σ_j, but this time party P_j sends σ_j not in response to a bilateral proof, but based on the current context of execution, as required by the protocol. In this case we just say "*P_i constructs certificate α_i for «some claim»* " and write

$$\alpha_i := \mathsf{certify}(\text{«}\textit{some claim}\text{»}) \; .$$

Note that in a concrete implementation the signed messages can be different from the actual claim being certified. For example, each player P_j could sign the message «*I have seen X_i*», and $n - t$ such signatures can be interpreted as a certificate for (iii).

To get some intuition why this approach fulfills the requirements, note that an adversary corrupting up to t players can never obtain sufficiently many signatures: In the method based on bilateral proofs an honest party never signs an incorrect claim, hence the adversary can collect at most $t < n - t$ signatures. Also, by the zero-knowledge property of bilateral proofs, no additional information is leaked. In the other case, when the construction of a certificate is protocol-driven, the soundness depends on the actual claim being certified, but it will be clear from the context.

In principle, for some claims a threshold smaller than $n - t$ could be sufficient to guarantee soundness, e.g. $t+1$ signatures would be sufficient for ensuring that example (i) is true. However, we choose the threshold $n - t$ to guarantee also *uniqueness* of the certified claims: since $n \geq 3t + 1$, a threshold of $n-t-1$ or less would be insufficient, as an adversary could easily obtain $n - t - 1$ signatures for two different claims. For example if $n = 3t + 1$ holds, an adversary could obtain from the honest parties t signatures for claim m, and t signatures for claim $m' \neq m$, which together with the signatures from the corrupted parties would yield $2t = n - t - 1$ signatures for m, and $2t$ signatures for m'. On the other hand, with a threshold of $n - t$ (and $n = 3t + 1$) at most one claim can be certified: If an adversary obtains $n - t = 2t + 1$ signatures for some claim m, then at least $t + 1$ of these signatures come from honest parties. Hence at most $n - t - (t + 1) = t$ honest parties will sign a different claim m', which means that the adversary obtains in total at most $2t$ signatures for m', which is less than the required threshold of $n - t$.

For simplicity, we use this basic signatures-based implementation of certificates in the MPC protocol presented in this section. Later, in Section 2.4 we describe an alternative implementation of certificates, based on *threshold* signatures, which results in a better efficiency of the entire protocol.

When presenting the protocols in the subsequent sections we keep in mind the above construction of certificates, but omit the details from the descriptions and use the shorthands $\mathsf{certify}(\dots)$ and $\mathsf{certify}_{\mathsf{zkp}}(\dots)$ instead. Naturally, we do include the corresponding sub-protocols and their bit complexities in the analysis of the proposed solution (cf. Section 2.3.8).

2.3.3 The circuit and the correctness invariant

To enable a precise description of the new protocol, we first formally model the circuit to be computed, and then give the invariant that is satisfied during the whole computation. For the clarity of presentation we assume in the following that every party provides exactly one input value and that there are only outputs which are to be disclosed to all parties (i.e., the final outputs are public). This is without loss of generality for the case with public outputs, and protocols for the general case with multiple input values can be derived by straightforward modifications (cf.Section 2.3.7). However, the issue of providing *private* outputs is more involved, and we discuss it in Section 2.7.

Circuit. We assume that the function to be computed is given as a circuit Circ over the plaintext space $\mathcal{M}$ of the homomorphic encryption scheme in use. The circuit is a set of labeled gates, where each label is a unique bitstring $G \in \{0,1\}^*$, used to identify the gate. The full description of a gate is given by a tuple $(G, \dots)$, consisting of a label and further parameters, depending on the type of the gate and its position in the circuit. We use $\mathcal{G}$ to denote the set of all gate labels of the circuit Circ. In the following we let $v : \mathcal{G} \to \mathcal{M} \cup \{\perp\}$ be a map from the labels into the the plaintext space, where $v(G)$ denotes the value of the gate G. Each gate $(G, \dots)$ can have one of the following types:

input gate: (G), consisting only of its label $G = (P_i, \mathsf{input})$, where $v(G)$ is equal to x_i, the input value provided by player P_i.

linear gate: $(G, \mathsf{linear}, a_0, a_1, G_1, \dots, a_l, G_l)$, where $l \geq 0$, $a_0, \dots, a_l \in \mathcal{M}$ are constants, and $v(G) = a_0 + \sum_{j=1}^{l} a_j \cdot v(G_j)$.

multiplication gate: $(G, \mathsf{mul}, G_1, G_2)$, where $v(G) = v(G_1) \cdot v(G_2)$.

output gate: $(G, \mathsf{output}, G_1)$, where $v(G) = v(G_1)$ is an output value of the circuit.

Dictionary. Throughout the computation each party P_i maintains a data structure containing the views of each party P_j on the intermediate values in the circuit. More precisely, P_i holds a *dictionary* Γ_i, which for each party P_j maps labels G to encryptions computed by the King P_j,

$$\Gamma_i : [n] \times \mathcal{G} \to \mathcal{C} \cup \{\perp\} \ .$$

Initially $\Gamma_i(j, G) = \perp$ for all labels and all $j \in [n]$. If $\Gamma_i(j, G) = X \neq \perp$, then from P_i's point of view gate G was completed by P_j, and X is a ciphertext encrypting the value $v(G)$. We say that X is the encryption of $v(G)$ *reported by* P_j *to* P_i, and that P_i *has accepted* X *from* P_j.

Correctness Invariant. The protocol guarantees, that if an honest party P_i accepts a ciphertext X reported by P_j, then X is an encryption of a correct value for gate G. Moreover, any two honest parties who accept an encryption of any party P_l for a gate G agree on the encryption. Formally, we have the following definition.

Definition 1 (Correctness Invariant) *The correctness invariant consists of the following four properties:*

1. (Agreement on input providers) *There exists a set $W \subseteq [n]$ with $|W| \geq n - t$, such that for every honest party P_i which has decided on a set of input providers $\{P_j : j \in W_i\}$, it holds that $W_i = W$.*

2. (Agreement on input encryptions) *For every two honest parties P_i, P_j, and for all $k, l, m \in [n]$ the following is true: if $\Gamma_i(k, (P_m, \mathsf{input})) \neq \perp$ and $\Gamma_j(l, (P_m, \mathsf{input})) \neq \perp$, then*

$$\Gamma_i(k, (P_m, \mathsf{input})) = \Gamma_j(l, (P_m, \mathsf{input})) =: X_m \ .$$

   Furthermore, if P_m is honest, then if $m \in W$ holds, $\mathscr{D}(X_m)$ is the initial input x_m of P_m, and if $m \notin W$ holds, then $\mathscr{D}(X_m)$ is equal to the default value for P_m's input.

3. (Correct gate encryption) *For every honest party P_i, if for any $G \in \mathcal{G}$ we have that $\Gamma_i(i, G) = X \neq \perp$, then $\mathscr{D}(X)$ is identical to the value of gate G obtained by decrypting the input encryptions held by P_i and evaluating the circuit on the plaintexts.*

4. (Agreement on encryptions of gates by the same king) *For every two honest parties P_i and P_j, for any king $P_k \in \mathcal{P}$ and any $G \in \mathcal{G}$, if $\Gamma_i(k, G) = X \neq \perp$ and $\Gamma_j(k, G) = X' \neq \perp$, then $X = X'$.*

This invariant is propagated from the initial input stage until the output stage is reached. Hence, a threshold decryption of the encrypted output value is guaranteed to yield correct computation results.

The basis of the correctness invariant. The correctness invariant is established in the input stage, which determines the values of all input gates. Due to the security properties of the input-stage protocol, these values are guaranteed to be *correct* in the sense that the each party providing input knows the actual value hidden in the encryption, and that this value is a valid input to the function to be computed.

2.3.4 Main protocol

The main protocol first invokes the input stage, then the evaluation stage, and finally the termination stage. At the end of the input stage parties agree on the set of parties that provide input to the computation, and on the (encrypted) values provided by these parties. In the evaluation stage the actual computation of the desired functionality takes place. For this purpose, every party starts one instance of the king protocol for the evaluation stage, and n instances of the slave protocol — one slave for every king. Each king runs for each gate G of the circuit the king subprotocol for G, and each slave runs the corresponding slave sub-protocol for G helping its king. Concurrently to these gate-protocols a termination protocol is executed, which allows all the parties to eventually terminate. Summing up, the top-level code for every player P_i with input x_i proceeds as follows:

1. Run the input stage code.
2. Wait until the input stage is completed, resulting in an initialized dictionary Γ_i. Then concurrently execute for each linear, multiplication, or output gate $G \in \mathcal{G}$ the corresponding protocols for computing G:
   - run the king protocol for G, to compute $\Gamma_i(i, G)$
   - for each $k \in [n]$ run the slave protocol for G, to help each king P_k in computation $\Gamma_k(k, G)$ (and so to compute $\Gamma_i(k, G)$)
3. Concurrently to the evaluation of the circuit run the termination code.

The corresponding sub-protocols are described in the next section.

Input stage code for P_i: given an input $x_i \in \mathcal{M}$ do the following:

1. compute $X_i := \mathscr{E}(x_i)$ and construct

$$\alpha_i := \mathsf{certify}_{\mathsf{zkp}}(\text{«}X_i \textit{ is } P_i\textit{'s valid input}\text{»}) \ .$$

   (every party P_j helps to construct at most one α_i, for each $P_i \in \mathcal{P}$)

2. enter execution of **`select`** protocol with input (X_i, α_i).
3. output value(s) returned by **`select`**.

Figure 2.1: The input stage code for P_i holding input $x_i \in \mathcal{M}$.

2.3.5 Sub-protocols used by the main protocol

To complete the presentation of the proposed MPC protocol we have to describe sub-protocols used at the various stages of the main protocol (cf. Sect. 2.3.4). Below we give intuitive descriptions of the sub-protocols, and present them in more detail in the corresponding figures.

2.3.5.1 Input stage

The goal of the input stage is to define an encryption of the input of each party. To ensure independence of the inputs, the parties are required to prove plaintext knowledge for their encryptions. In a synchronous network we could simply let the parties broadcast their encryptions. However, in an asynchronous setting with an active adversary we cannot guarantee that each party contributes an input value, since it is impossible to distinguish between a slow honest party and a corrupted party not sending anything. Therefore a protocol is used which selects inputs from at least $(n-t)$ so-called *input providers*. These are the parties, whose private inputs will be used in the actual computation of the circuit, and for the remaining inputs the default values will be used. The protocol will guarantee, that all honest parties obtain the correct inputs of input providers, and so will be able to perform the computation.

The input stage consists of two main steps, as shown in Fig. 2.1. In the first step, each party P_i encrypts its input value x_i to obtain a ciphertext X_i, and constructs a certificate $\alpha_i = \mathsf{certify}_{\mathsf{zkp}}(\text{«}X_i \textit{ is } P_i\textit{'s valid input}\text{»})$, using bilateral zero-knowledge proofs and digital signatures. The certificate α_i certifies that X_i is a valid value to be used for **`select`** protocol (see below). In particular, it implies that P_i knows the encrypted value, that the encrypted value is from the valid range, and that X_i is P_i's unique possible input encryption to the circuit.

In the second step, parties run **`select`** protocol which allows them to select the inputs of at least $n-t$ players, and guarantees agreement on the selected values. Finally, the players output the values returned by the execution of **`select`** protocol.

Before we argue about the security and correctness of the input stage, we describe **`select`** in more detail. Since we are going to use this protocol also in other situations, we present it as a stand-alone protocol with appropriate parameters. In general, **`select`** is parametrized by a condition φ, which has to be satisfied for each input to the protocol, and certified by an appropriate certificate. For example, in the input stage the condition for P_i's input X_i is «*X_i is P_i's valid input*», and the corresponding certificate α_i was generated in the first step, as described above. We require that *φ implies uniqueness*, i.e., that every party can obtain a corresponding certificate valid for φ for at most one input value used in any execution of **`select`**.

The protocol proceeds as follows: First P_i distributes its input (X_i, α) to all parties, and then constructs and distributes a *certificate of distribution β_i*, which proves that P_i has distributed (X_i, α_i) to at least $n-t$ parties. When a party collects $n-t$ certificates of distribution, she knows that at least $n-t$ parties have their certified inputs distributed to at least $n-t$ parties. So, at least $n-t$ parties had their certified inputs distributed to at least $(n-t)-t \geq t+1$ *honest* parties. Hence, if all honest parties echo the certified inputs they saw and collect $n-t$ echoes, then all honest parties will end up holding the certified input of the $n-t$ parties, which had their certified inputs distributed to at least $t+1$ honest parties. These $n-t$ parties will eventually be the input providers. To determine who they are, n Byzantine Agreements are run. The protocol for selecting input providers is given in more detail in Fig. 2.2. In this protocol P_i keeps track of arriving inputs and certificates with help of three auxiliary sets $A_i, \boldsymbol{\mathcal{A}}_i, C_i$, where A_i collects the indexes of the parties from which P_i has received certified inputs, $\boldsymbol{\mathcal{A}}_i$ holds the corresponding inputs and certificates (valid for the condition $\varphi(i)$), and C_i collects the indexes of the parties from which P_i has received certificates of distribution.

Protocol **`select`**(φ)*, code for* P_i: given input X_i with a certificate α_i valid for condition $\varphi(i)$ initialize sets $A_i, \boldsymbol{\mathcal{A}}_i, C_i$ as empty, then execute the following rules concurrently:

DISTRIBUTION:

1. send (X_i, α_i) to all parties.
2. construct and send to all parties $\beta_i :=$ certify(«*we hold* P_i*'s input* X_i»)

GRANT CERTIFICATE OF DISTRIBUTION:

1. upon first (X_j, α_j) from P_j with α_j valid for $\varphi(j)$: add j to A_i, add (X_j, α_j) to $\boldsymbol{\mathcal{A}}_i$, and send $\sigma_i := \mathsf{Sign}_i$(«*we hold* P_j*'s input* X_j») to P_j.

ECHO CERTIFICATE OF DISTRIBUTION:

1. upon (X_j, β_j) with β_j valid for «*we hold* P_j*'s input* X_j» and $j \notin C_i$: add j to C_i and send (X_j, β_j) to all parties.

SELECTION:

If $|C_i| \geq n-t$, stop executing all above rules and proceed as follows:

1. send $(A_i, \boldsymbol{\mathcal{A}}_i)$ to all parties.
2. collect a set $\{(A_j, \boldsymbol{\mathcal{A}}_j)\}_{j \in J}$ of $(n-t)$ well-formed $(A_j, \boldsymbol{\mathcal{A}}_j)$; let $B_i := \bigcup_{j \in J} A_j$ and $\boldsymbol{\mathcal{B}}_i := \bigcup_{j \in J} \boldsymbol{\mathcal{A}}_j$
3. enter n Byzantine Agreements (BAs) with inputs $v_1, \ldots, v_n \in \{0, 1\}$, where $v_j = 1$ iff $j \in B_i$.
4. let $w_1, \ldots, w_n$ be the outputs of the BAs; let $W := \{j \in [n] | \; w_j = 1\}$.
5. $\forall\, j \in B_i \cap W$: send $(X_j, \alpha_j) \in \boldsymbol{\mathcal{B}}_i$ to all parties.
6. collect and output $(W, \{(X_j, \alpha_j)\}_{j \in W})$.

Figure 2.2: Protocol **`select`**(φ): code for P_i holding input (X_i, α_i), where α_i certifies that X_i satisfies $\varphi(i)$.

Lemma 1 (Properties of `select`(φ)) *If there are at most $t < n/3$ corrupted parties and every honest party P_i enters* **`select`**(φ) *with input (X_i, α_i), where φ implies uniqueness and α_i is valid for $\varphi(i)$, then the following conditions are satisfied:*

Termination: All honest parties eventually terminate the protocol.

Agreement on output: There exists a set $W \subseteq [n]$ with $|W| \geq n - t$, such that every honest party P_i outputs $(W, \{(X_j, \alpha_j)\}_{j \in W})$, where each α_j is valid for $\varphi(j)$.

Proof. *(sketch)* As the first step in the proof we argue that if all honest parties start running the protocol in Fig. 2.2, then some honest party will eventually trigger the rule SELECTION. Consider namely a dead-locked execution where this did not happen. In such an execution all honest parties are still executing all rules. So, by the rule GRANT CERTIFICATE OF DISTRIBUTION all honest parties complete Step 2 in the rule DISTRIBUTION. So, by the rule ECHO CERTIFICATE OF DISTRIBUTION, the set C_i eventually grew to size $n - t$ at all honest parties, and so the rule SELECTION was eventually triggered by all honest parties, a contradiction.

Now consider any execution where at least one honest party, P_i, triggered SELECTION. This party has $|C_i| \geq n - t$, and therefore has sent at least $n - t$ certificates β_j to all other parties in the rule ECHO CERTIFICATE OF DISTRIBUTION. Therefore all honest parties P_l will eventually have $|C_l| \geq n - t$ and trigger SELECTION. It is then clear that all honest parties will eventually reach Step 6 in SELECTION and start waiting for (X_j, α_j) for each $j \in W$. So to prove *Termination* it is sufficient to argue that all (X_j, α_j) eventually arrive in Step 6. To see this, observe that if $j \in W$ holds, then at least one honest party P_l had $j \in B_l$ in Step 3. This P_i will eventually reach Step 5 and send (X_j, α_j) to all parties, and (X_j, α_j) will thus eventually arrive.

Now we argue *Agreement on output*. Let W to be the set defined in Step 4 by the first honest party. The agreement on W follows from the fact that W is defined by running Byzantine Agreements. It remains to argue that $|W| \geq n - t$. Since $|C_i| \geq n - t$ for all honest parties P_i reaching Step 4 in SELECTION, it is sufficient to show a stronger claim, that $(\bigcup_{i \in H} C_i) \subseteq W$, where H denotes the set of honest parties reaching Step 4. To see this, notice that if $j \in C_i$, then P_i saw β_j valid for «*we hold P_j's input X_j*». This means that at least $(n - t) - t \geq t + 1$ *honest* parties P_l issued a signature on «*we hold P_j's input X_j*» in GRANT CERTIFICATE OF DISTRIBUTION, and added (X_j, α_j) to $\boldsymbol{\mathcal{A}}_l$. This was done

```
Slave $P_i$ helping king $P_k$ in evaluation of $(G, \mathsf{linear}, a_0, a_1, G_1, \ldots, a_l, G_l)$:

  1. wait until $\Gamma_i(k, G_u) \neq \perp$, for all $u = 1 \ldots l$.
  2. compute $\Gamma_i(k, G) := A_0 \oplus \left(\bigoplus_{u=1}^{l} (a_u \star \Gamma_i(k, G_u))\right)$ .
```

Figure 2.3: Code for P_i helping king P_k to evaluate a linear gate.

before P_l reached Step 1 in SELECTION (at which point P_l stops executing rules other than SELECTION). So, (X_j, α_j) is in the set $\boldsymbol{\mathcal{A}}_l$ of at least $t+1$ honest parties at Step 1 in SELECTION. Since $(t+1)+(n-t) > n$, this means that every honest party P_m reaching Step 3 will have (X_j, α_j) in $\boldsymbol{\mathcal{B}}_l$ and will enter the Byzantine Agreement with $v_j = 1$. Hence, all parties will have $w_j = 1$ and so $j \in W$ holds, as desired.

Finally, we prove that honest parties agree also on the values in set $\{(X_j, \alpha_j)\}_{j \in W}$. If P_i's output contains (X_j, α_j), then α_j is valid for $\varphi(j)$. Since the parties agree on W, it is enough to argue that there exists a unique such value (X_j, α_j). Clearly, this follows from the assumption that $\varphi(j)$ implies uniqueness. □

Since the output of the input stage is just the output of **`select`** protocol, Lemma 1 implies directly that after termination of the input stage the first two properties of the correctness invariant, i.e. agreement on circuit and agreement on input encryptions, are satisfied. The remaining two properties of the invariant are satisfied as well, but before we show it, we complete the presentation of the MPC protocol by describing the required sub-protocols.

2.3.5.2 Computing linear gates

Due to the homomorphic property of encryption, linear gates can be computed locally, without interaction. That is, if a slave P_i has accepted P_k's encryptions of inputs to a linear gate $(G, \mathsf{linear}, a_0, G_1, a_1, \ldots, G_l, a_l)$, i.e. when $\Gamma_i(k, G_u) \neq \perp$, for $u = 1 \ldots l$, then P_i computes locally $\Gamma_i(k, G) := A_0 \oplus \left(\bigoplus_{u=1}^{l} (a_j \star \Gamma_i(k, G_u))\right)$, where A_0 is a "dummy" encryption of a_0, computed using a fixed, publicly-known bit-string as randomness. The code for slave P_i helping king P_k is summarized in Fig. 2.3. The code for the king is empty in the case of linear gates, since the slave code run by P_i helping king P_i (i.e., for $i = k$) performs all the necessary computation.

```
King $P_k$ evaluating a multiplication gate $(G, \mathsf{mul}, G_1, G_2)$:
  1. wait until $\Gamma_k(k, G_1) = C_1 \neq \perp$ and $\Gamma_k(k, G_2) = C_2 \neq \perp$
  2. generate a randomizer $(R, U, \alpha)$ for $G$, and send it to all parties:
     (a) collect a set $S := \{(R_j, U_j, \sigma_j, \beta_j)\}_{j \in J}$, with $|J| \geq t+1$, where
         $\sigma_j = \mathsf{Sign}_j(«(R_j, U_j)$ : part of $P_k$'s randomizer for $G»)$
         $\beta_j = \mathsf{certify}_{\mathsf{zkp}}(«P_j$ knows $r_j$ in $R_j$; $U_j$ is a randomization of $r_j \star C_1»)$
     (b) send $S$ to all parties
     (c) compute $R := \bigoplus_{j \in J} R_j$, and $U := \bigoplus_{j \in J} U_j$
     (d) construct $\alpha := \mathsf{certify}(«(R, U)$ : $P_k$'s randomizer for $G»)$
  3. collect a set $V = \{(z_j, \gamma_j)\}_{j \in J'}$, with $|J'| \geq t_{\mathsf{D}}$, where each $z_j$ is $P_j$'s
     decryption share for $Z = C_2 \oplus R$, and $\gamma_j$ certifies $z_j$'s validity
  4. send $V$ to all parties
```

Figure 2.4: Code for king P_k evaluating a multiplication gate.

2.3.5.3 Computing multiplication gates

The protocol for computation of multiplication gates is more involved. Roughly speaking, the idea is that each king P_k leads the computation of the encrypted product in *his* instance of the evaluation protocol for each particular multiplication gate. That is, given a gate $(G, \mathsf{mul}, G_1, G_2)$ such that $\Gamma_k(k, G) = \perp$, $\Gamma_k(k, G_1) = C_1$, and $\Gamma_k(k, G_2) = C_2$, with $C_1, C_2 \neq \perp$, the king P_k leads all the players (who act as *slaves*) through the following computation.

Let c_1, c_2 denote the values hidden in the ciphertexts C_1, C_2, respectively. First a randomizer (R, U, α) is generated, where R is a threshold encryption of a random element $r \in \mathcal{M}$ (unknown to the parties and the adversary), $U = \mathscr{R}(r \star C_1)$, i.e., U is a random threshold encryption of rc_1, and α is a certificate of the encryptions' correctness. Then P_k sends the randomizer to all parties, and waits until the parties answer with decryption shares of the ciphertext $Z = C_2 \oplus R$, which is an encryption of $z = c_2 + r$. Once sufficiently many (i.e., at least t_{D}) decryption shares arrive, P_k sends them to all parties, which allows each P_i to decrypt z, and compute an encryption of the product c_1c_2, using the homomorphic

```
Slave $P_i$ helping king $P_k$ in evaluation of $(G, \mathsf{mul}, G_1, G_2)$:

  1. wait until $\Gamma_i(k, G_1) = C_1 \neq \perp$ and $\Gamma_i(k, G_2) = C_2 \neq \perp$

  2. help to generate a randomizer $(R, U, \alpha)$ for $G$:

     (a) pick a random value $r_i \in \mathcal{M}$, then compute and send to king $P_k$
         the tuple $(R_i, U_i, \sigma_i, \beta_i)$, where $R_i := \mathscr{E}(r_i)$,  $U_i := \mathscr{R}(r_i \star C_1)$,
         $\sigma_i := \mathsf{Sign}_i(«(R_i, U_i)$ : part of $P_k$'s randomizer for $G»)$, and
         $\beta_i := \mathsf{certify}_{\mathsf{zkp}}(«P_i$ knows $r_i$ in $R_i$;  $U_i$ is a randomization of $r_i \star C_1»)$
     (b) wait for set $S = \{(R_j, U_j, \sigma_j, \beta_j)\}_{j \in J}$ from $P_k$, with $|I| \geq t+1$
     (c) compute $R := \bigoplus_{j \in J} R_j$, and $U := \bigoplus_{j \in J} U_j$
     (d) compute $\rho_i := \mathsf{Sign}_i(«(R, U) : P_k$'s randomizer for $G»)$;  send $\rho_i$ to $P_k$

  3. wait for $(R, U, \alpha)$ from $P_k$, with $\alpha$ valid for $«(R, U) : P_k$'s randomizer for $G»$;
     compute and send to $P_k$ tuple $(z_i, \gamma_i)$, where $z_i$ is $P_i$'s decryption
     share for $Z := C_2 \oplus R$, and $\gamma_i := \mathsf{certify}_{\mathsf{zkp}}(«z_i$ is valid$»)$

  4. wait for decryption shares $V$ from $P_k$, with $|V| \geq t_{\mathsf{D}}$

  5. decrypt $z := \mathscr{D}(Z, V)$ and compute $\Gamma_i(k, G) := (z \star C_1) \ominus U$
```

Figure 2.5: Code for slave P_i helping king P_k to evaluate a multiplication gate.

property of the encryption, and the fact that $c_1 c_2 = (c_2 + r)c_1 - rc_1$. That is, P_i computes $\Gamma_i(k, G) := (z \star C_1) \ominus U$.

A detailed description of the multiplication procedure for a king and for a slave is given in Figures 2.4 and 2.5, respectively. Note that when computing a certificate β_i for the claim

$$«P_i \textit{ knows } r_i \textit{ in } R_i; \;\; U_i \textit{ is a randomization of } r_i \star C_1»$$

the variables P_i, R_i, U_i, and C_1 are replaced by the actual values they stand for (as it is also the case for all the variables in all other claims in this protocol), while r_i stays in the claim as a literal, since it is just a name for the plaintext hidden in the ciphertext R_i.

```
King $P_k$ evaluating an output gate $(G, \mathsf{output}, G_1)$:
  1. wait until $\Gamma_k(k, G_1) = C \neq \perp$
  2. collect a set $T = \{(c_j, \delta_j)\}$ of $t_\mathsf{D}$ decryption shares for $C$,
     with corresponding validity certificates $\delta_j$
  3. compute $c := \mathscr{D}(C, T)$; send $T$ to all parties
  4. construct and send to all parties $\zeta_G = \mathsf{certify}(«P_k\text{'s value of } G \text{ is } c»)$
```

Figure 2.6: Code for king P_k evaluating an output gate.

```
Slave $P_i$ helping king $P_k$ in evaluation of $(G, \mathsf{output}, G_1)$:
  1. wait until $\Gamma_i(k, G_1) = C \neq \perp$
  2. compute a decryption share $c_i := \mathscr{D}_i(C)$ and a certificate
     $\delta_i := \mathsf{certify}_\mathsf{zkp}(«c_i \text{ is valid}»)$; send $(c_i, \delta_i)$ to $P_k$
  3. wait for decryption shares $T = \{(c_j, \delta_j)\}$ from $P_k$;
     compute $c := \mathscr{D}(C, T)$,
  4. help $P_k$ to compute a certificate $\zeta_G$ valid for $«P_k\text{'s value of } G \text{ is } c»$;
     wait for $\zeta_G$ from $P_k$
  5. set $v(G) = c$; mark $G$ as decrypted
```

Figure 2.7: Code for P_i helping P_k to evaluate an output gate.

2.3.5.4 Output stage

When P_i notices that the computation of an output gate $(G, \mathsf{output}, G_1)$ is completed by some king P_k (i.e. $\Gamma_i(k, G) = C \neq \perp$), but the gate has not been decrypted so far, then P_i sends a decryption share c_i of C to P_k along with a certificate that the decryption share is correct. King P_k collects sufficiently many certified decryption shares, and sends them to all the slaves. Every slave P_i decrypts the output using the received decryption shares. Subsequently the parties construct a certificate ζ_G, which certifies that the decrypted output value is correct. With such a certificate any party P_i can convince any other party about the correctness of the output. As we describe in the next section, this property will be useful in the termination protocol.

During the protocol each party executes concurrently the following rules, where $\mathcal{G}_O$ denotes the set of all output gates:

RULE 1:

```
1. wait until every output gate $G \in \mathcal{G}_O$ is marked decrypted
2. send to all parties $\{(G, v(G), \zeta_G)\}_{G \in \mathcal{G}_O}$, where every $\zeta_G$ is a
   certificate valid for «$P_j$'s value of $G$ is $v(G)$» for some $j \in [n]$
3. terminate
```

RULE 2:

```
1. wait for a message $\{(G, v(G), \zeta_G)\}_{G \in \mathcal{G}_O}$, where every $\zeta_G$ is a
   certificate valid for «$P_j$'s value of $G$ is $v(G)$» for some $j \in [n]$
2. mark every output gate $G \in \mathcal{G}_O$ as decrypted
```

Figure 2.8: The code for terminating P_i.

2.3.5.5 Termination

As described above each king will eventually learn the value of the output gate. However, to guarantee that every king completes the evaluation led by him, it is necessary that the slaves run by every party P_i keep running even after the king run by P_i has finished and learned the output values. To allow to terminate also the slaves, the parties execute a termination protocol. In fact, the termination protocol allows a party P_i to terminate even its king process before completing the entire circuit, for example when some other king has completed the computation and P_i has received all the outputs with the corresponding certificates.

More precisely, when a party P_i collects certified output values for all output gates (cf. previous section), either through its own king process, or as a slave from the other kings, or directly from some other party as a part of the termination code, then P_i is ready for terminating. For this purpose P_i sends to all parties the complete set of collected certified output values, and terminates. This echoing of output values ensures that every honest party eventually receives all output values and terminates. The code for terminating P_i is summarized in Fig. 2.8.

2.3.6 Security analysis

Our protocol can be proved secure in the model described in Section 2.1. A formal proof that the protocol can be simulated can be given along the lines of the proof in [CDN01], using two helping lemmas. In this section, we first present the helping lemmas and sketch their proofs, and then we discuss how the lemmas allow to give a proof along the lines of [CDN01]. We stress, that the proposed constructions are in the plain model, without making use of the random-oracle methodology.

2.3.6.1 Helping lemmas

Lemma 2 (The correctness invariant) *If there are at most $t < n/3$ corrupted parties, then the correctness invariant (Definition 1) is satisfied at any point in the protocol.*

Proof. *(sketch)* Recall the properties of the correctness invariant (cf. Def. 1):

1. *Agreement on input providers.*
2. *Agreement on input encryptions.*
3. *Correct gate encryption.*
4. *Agreement on encryptions of gates by the same king.*

As mentioned previously, the first two properties follow directly from the *agreement on output* of **`select`**(φ) (cf. Lemma 1). More precisely, until the first honest party reaches Step 4 in SELECTION in Fig. 2.2 we can take W to be any subset of size $n-t$. After that we take W to be the set defined in Step 4 by the first honest party. As shown in proof of Lemma 1, it holds that $|W| \geq n-t$ and all honest parties agree on W and on $\{(X_j, \alpha_j)\}_{j \in W}$, as required. Furthermore, the agreement on input encryptions follows from the fact that the corresponding condition implies uniqueness, since during the input stage (Fig. 2.1) each honest party helps to construct at most one α_i for each $i \in [n]$, i.e issues at most one signature on a message of the form « $\cdot$ *is P_i's valid input*». To construct two different certificates valid for statements of the form « $\cdot$ *is P_i's valid input*» a total of $2(n-t)$ signatures are needed. So, to certify two different messages at least $2(n-t) - n = n - 2t \geq t + 1$ parties must sign more than one message of this form. If there are at most t corrupted parties and honest parties sign only one message of the specified form, clearly at most one value is certified.

We now argue that the third property, *correct gate encryption*, is an invariant. Clearly it holds for all input gates of P_k. Furthermore, if it holds for the encryptions $\Gamma_i(k, G_u)$ for $u = 1, \ldots, l$ in Step 1 of the protocol for evaluating linear gates (Fig. 2.3), then it clearly holds for the result $\Gamma_i(k, G)$, by the homomorphic properties of the encryption scheme. We then consider the multiplication protocol (Fig. 2.4 and 2.5). Assume first that it holds for (R, U) in Step 2(c) that $\mathscr{D}(U) = \mathscr{D}(R) \cdot \mathscr{D}(C_1)$, and that Step 4 in slave's protocol terminates. In that case, by the homomorphic properties, $z = \mathscr{D}(C_2) + \mathscr{D}(R)$, and thus

$$\mathscr{D}(\Gamma_i(k, G)) = [\mathscr{D}(C_2) + \mathscr{D}(R)] \cdot \mathscr{D}(C_1) - \mathscr{D}(R) \cdot \mathscr{D}(C_1) = \mathscr{D}(C_1) \cdot \mathscr{D}(C_2),$$

as desired. Therefore it is sufficient to argue that $\mathscr{D}(U) = \mathscr{D}(R) \cdot \mathscr{D}(C_1)$. This follows directly from the distributive law in the plaintext space $\mathcal{M}$, and from the fact that the certificates β_i in Step 2(a) guarantee that (except with negligible probability) $\mathscr{D}(U_i) = \mathscr{D}(R_i) \cdot \mathscr{D}(C_1)$, for all $i \in I$.

Finally we argue that the last property, *agreement on encryptions of gates by the same king*, is an invariant. We proceed by induction. For input encryptions, it follows from the second property. Furthermore, it is clearly preserved by linear gates. Hence it remains to consider the protocol for multiplication gates. Assume that P_i and P_j agree on the inputs of P_k to a multiplication, i.e. that in Step 1 of the multiplication protocol (Fig. 2.5) the following holds

$$\Gamma_i(k, G_1) = \Gamma_j(k, G_1) \neq \perp \quad \text{and} \quad \Gamma_i(k, G_2) = \Gamma_j(k, G_2) \neq \perp \ .$$

We have to argue that if $\Gamma_i(k, G)$ and $\Gamma_j(k, G)$ become defined in Step 5, then $\Gamma_i(k, G) = \Gamma_j(k, G)$ holds. Since $\star$ and $\ominus$ are functions, and since P_i and P_j compute the gate's output $\Gamma_i(k, G)$ resp. $\Gamma_j(k, G)$ as $(z \star C_1) \ominus U$, it is sufficient to demonstrate that P_i and P_j agree on z, C_1 and U. This in turn follows from the agreement on Z, C_1 and U, since the decryption in Step 5 is correct (except with a negligible probability).[3] Since $\oplus$ is a function and Z is equal to $C_2 \oplus R$, it is enough to argue the agreement on C_2, R, C_1 and U. We have an agreement on C_1 and C_2 by assumption. To see that P_i and P_j must agree on (R, U) observe that each of them saw in Step 3 a certificate α valid for a message of the form «$(\cdot, \cdot)$: P_k*'s randomizer for* G». By an argument like the one used for the second property, it follows that α guarantees uniqueness, so there is at most one certified message of this form, hence P_i and P_j have the same certified randomizer (R, U). □

[3] Recall that the decryption shares sent in V come along with correctness certificates.

Lemma 3 (Termination) *If there are at most $t < n/3$ corrupted parties and all honest parties start running the protocol, then all honest parties will eventually terminate the protocol.*

Proof. *(sketch)* It is clear that the first step of the input stage protocol (Fig. 2.1) eventually terminates. Hence by the *termination* property of **`select`** (Lemma 1) it follows that if all honest parties start the input stage, then all honest parties eventually terminate this stage.

Assume first that at least one honest party, say P_i, terminates, i.e. reaches Step 3 in RULE 1 in Fig. 2.8. This means that P_i has obtained a certified output value for every output gate in the circuit, i.e. for every output gate G party P_i has a tuple $(G, v(G), \zeta_G)$ with ζ_G valid for «P_j*'s value of* G *is* $v(G)$», for some party $P_j \in \mathcal{P}$. Note that it doesn't matter, which party P_j was the king computing the certificate for an output value $v(G)$: by the correctness invariant, and by the soundness of the certificates, any party P_j can obtain a certificate only for the unique value of $v(G)$ implied by the inputs.

Since just before terminating P_i forwards all these tuples to all parties, every honest party eventually receives all certified output values, and becomes ready for termination. We have hence established, that if at least one honest party terminates, then all honest parties terminate. To complete the argument, we have to show that at least one honest party indeed terminates.

Assume now that all parties terminated the input stage, that no party reached Step 3 in RULE 1 in Fig. 2.8, and that the protocol is dead-locked. This means that from the point where the input stage terminated and throughout the execution each party was running for each gate a copy of the king code and a n copies of the slave code. Since no king or slave ever waits for more than $n-t$ parties, and there are $n-t$ honest parties (none of which terminated, by assumption), this guarantees that no honest party dead-locked in a king or slave code for any gate. In particular, all honest parties reached Step 4 in king's output gate protocol (Fig. 2.6), for every output gate G. This implies that all $n - t$ honest parties have sent the certified output values to all parties, which in turn implies that every honest party eventually obtains a certified value of every output gate. But this means that every honest party eventually reaches Step 3 in RULE 1 in Fig. 2.8. A contradiction. □

2.3.6.2 Security argument

As mentioned previously, the proof of security of our protocol goes along the lines of [CDN01]. Below we sketch the main argument, highlighting the points where Lemmas 2 and 3 are used.

By Property 1 (*agreement on input providers*) a set of at least $n-t$ parties have their inputs considered, as required by the model. Furthermore, by Property 3 (*correct gate encryption*), the output c, obtained by P_i when decrypting the output ciphertext in Step 3 of the output-gate code, will be correctly defined from the plaintexts of the input ciphertexts held by P_i. Since all honest parties agree on the input ciphertexts (Property 2), all honest parties P_i will agree on the output c in Step 3 of the output-gate code. By the soundness of the certificates and the fact that they guarantee uniqueness, this implies that all honest parties terminate the protocol in Fig. 2.8 with the output being the common value $v(G) = c$, as no other value can get a suitable certificate when there are at most t corrupted parties. Since c is the result of evaluating the circuit on the plaintexts of the input ciphertexts and, by Property 2, the input ciphertext X_l of honest party P_l contains the correct input x_l, the result c can indeed be obtained by restricting the set of input providers to a set of size at least $n - t$ and then changing only the inputs of the corrupted parties.

The privacy of the protocol (formally defined by the simulator only being given the inputs of the corrupted parties in the simulation) follows mainly from the fact that all inputs are encrypted using a semantically secure encryption scheme and that all proofs are zero-knowledge. So, the only knowledge leaked about the inputs of the honest parties is through decryptions of ciphertexts.

The decryptions take place only in Step 5 of multiplication-gate code (Fig. 2.5), and in Step 3 of output-gate code (Figs. 2.6 and 2.7). By the correctness of the protocol the knowledge leaked in Step 3 of output-gate is the result of the computation, which is allowed to leak by the model. So it remains to argue that no knowledge is leaked in Step 5 of multiplication. To see this, observe that the value revealed by the decryption is $z = c_2 + \sum_{j \in J} r_j$, which has a potential of leaking knowledge about c_2 (which possibly is to be kept secret). Since each term r_j from an honest party is chosen uniformly at random and all r_j are chosen independently (this is the purpose of having all parties, in particular the corrupted parties, prove plaintext knowledge of their r_i in Step 2(a)), it is sufficient to show that each revealed value $z = c_2 + \sum_{j \in J} r_j$ contains at least one honest value r_j *which did not enter another revealed value.*

First observe that since $|J| \geq t+1$, at least one r_j came from an honest party. Observe also that each of the randomizers r_j is associated uniquely to one (P_k, G) by the signature σ_j (issued in Step 2(a) and checked in Step 2(c)). Therefore r_j only enters values $z = c_2 + \sum_{j \in J} r_j$ leaked in decryptions in Step 5 of the multiplication-gate protocol for the specific (P_k, G) in consideration. It is therefore sufficient to show that for each (P_k, G) there is only one value z for which knowledge is leaked. This follows from the uniqueness guaranteed by the certificates α and from Lemma 2. More precisely, by the uniqueness of α there exists at most one value (R, U) with a certificate valid for «(R, U) : *P_k's randomizer for G*». Furthermore, since the honest parties agree on the gate encryptions of P_k (cf. Lemma 2, Property 4), there exists at most one value $Z = C_2 \oplus R$ for which honest parties issue decryption shares in Step 3 for a given choice of (P_k, G). Therefore each value $z = c_2 + \sum_{j \in J} r_j$ on which knowledge is leaked through decryption shares from honest parties, at least one r_i came from an honest party and did not enter another value on which knowledge was leaked, as desired.

2.3.7 Circuits with multiple inputs

So far we assumed that every party provides exactly one input value. This is essentially without loss of generality, since the modifications necessary for the general case with multiple inputs are straightforward, as described below.

If a party P_i has $\ell > 1$ input values, $x_{i,1}, \ldots, x_{i,\ell}$, then in the input stage (Fig. 2.1) P_i computes corresponding ciphertexts $X_{i,1}, \ldots, X_{i,\ell}$, and using bilateral zero-knowledge proofs obtains a certificate α_i certifying the correctness and knowledge of all corresponding plaintexts. Then P_i defines its input for the **`select`** protocol as $(X_i = (X_{i,1}, \ldots, X_{i,\ell}), \alpha_i)$ and proceeds as previously.

2.3.8 Efficiency analysis

In this section we consider the communication complexity of the protocol. We omit computational complexity from the analysis, since it is clearly polynomial, and the bottleneck of distributed computing is in the communication overhead. Moreover, since it is not hard to see that the round complexity is linear in the depth of the circuit, in the rest of this section we focus on bit complexity, i.e. the number of bits communicated between the parties.

Conventions and notation. We assume that all encryptions, all signature shares, all signatures and all pairwise proofs communicate $\mathcal{O}(\kappa)$ bits, and that the bit complexity of a Byzantine Agreement is $\mathcal{O}(n^2\kappa)$ [CKS00]. For completeness, we consider the case with multiple inputs per party and multiple public outputs. We use c_I, c_M, and c_O to denote the total number of input-, multiplication- and output gates, respectively. Moreover, we use π to denote the size of a certificate constructed by $\mathsf{certify}_{\mathsf{zkp}}(\cdot)$ or $\mathsf{certify}(\cdot)$. Since a certificate consists just of $n-t$ signatures, each of size $\mathcal{O}(\kappa)$, we have that $\pi = \mathcal{O}(n\kappa)$.

Input stage. In the input protocol (Fig. 2.1) for every input value two certificates are constructed and echoed by all parties, causing $\mathcal{O}(n^2\pi)$ bits of communication. Additionally, the echoing of the encrypted values costs $\mathcal{O}(n^2\kappa)$ bits. Finally, the n Byzantine Agreements executed during the input stage cause an overall of $\mathcal{O}(n^3\kappa)$ bits of communication. Therefore the total bit complexity of the input stage is bounded by $\mathcal{O}(c_I(n^2\pi + n^2\kappa) + n^3\kappa)$.

Evaluation stage. In the king's protocol the dominating values sent are the sets S and V in the multiplication protocol (Fig. 2.4), and the set T in the output-gate protocol (Fig. 2.6). These sets have size $\mathcal{O}(n\kappa + n\pi)$ and are sent to all n slaves. This gives a communication complexity of $\mathcal{O}(n(n\kappa + n\pi))$ each time a set is sent, which results in a bit complexity of $\mathcal{O}((c_M + c_O)n(n\kappa + n\pi))$ for running the protocols in Figs. 2.4 and 2.6. These protocols are run by all n kings, yielding a total bit complexity of $\mathcal{O}((c_M + c_O)n^2(n\kappa + n\pi))$.

In the slave's protocol the construction and sending of the values in Steps 2(a) and 3 of the multiplication gate (Fig. 2.5), and in Step 2 of the output gate protocol (Fig. 2.7) all use $\mathcal{O}(n\kappa + \pi)$ bits of communication, and these are the dominating instructions. Each construction of a certificate is done at most once for each gate for each king being helped. This yields a total of $\mathcal{O}((c_M + c_O)n(n\kappa + \pi))$ bits for running the slave's protocol. Since the protocol is run by all n slaves, this yields a total bit complexity of $\mathcal{O}((c_M + c_O)n^2(n\kappa + \pi))$.

Termination stage. It is easy to verify that the total bit complexity of the terminating protocol in Fig. 2.8 is $\mathcal{O}(c_O n^2(\pi + \kappa))$.

Total communication. Summing all terms we get bit complexity of

$$\mathcal{O}(c_I(n^2\pi + n^2\kappa) + c_O n^2\pi + (c_M + c_O)n^2(n\kappa + n\pi) + n^3\kappa). \tag{2.1}$$

Using $\pi = \mathcal{O}(n\kappa)$ bits, and assuming that $c_O \geq 1$ holds, we obtain, that the total bit complexity of the proposed MPC protocol is bounded by $\mathcal{O}(c_I n^3\kappa + (c_M + c_O)n^4\kappa)$. That is, $\mathcal{O}(n^4\kappa)$ bits are communicated per multiplication or output gate, and $\mathcal{O}(n^3\kappa)$ bits per input gate.

2.4 Better efficiency via threshold signatures

A closer look at the bit complexity (2.1) of the protocol from the previous section shows that the dominating factor is determined by the cost of sending certificates. Since a certificate for «*some claim*» is a collection of sufficiently many (i.e., at least $n - t$) signatures $\sigma_i = \mathsf{Sign}_i(\text{«}\textit{some claim}\text{»})$, the size of one certificate is $\mathcal{O}(n\kappa)$ bits. Here we describe an alternative construction of certificates, which results in certificates of size $\mathcal{O}(\kappa)$ bits only. Plugging-in this new construction into the MPC protocol yields a protocol with the overall bit complexity reduced by a factor of n.

To create short certificates we employ a *threshold* signature scheme $(\mathscr{S}, \mathscr{V})$ with a threshold $t_\mathsf{S} = n - t$, in which the ability to compute signatures is shared among the parties (cf. Sect. 2.2.3). To construct a certificate α valid for «*some claim*», i.e. a threshold signature for «*some claim*», a party must collect t_S correct *signature shares* from different parties, where each share is computed as $\sigma_j = \mathscr{S}_j(\text{«}\textit{some claim}\text{»})$. Then the signature shares are combined[4] to a single signature using the algorithm $\mathscr{S}$:

$$\alpha := \mathscr{S}(\text{«}\textit{some claim}\text{»}, \{\sigma_j\}_{j \in J}), \text{ where } |J| \geq t_\mathsf{S}\,.$$

Any party knowing the corresponding public verification key v can verify correctness of the resulting signature using algorithm $\mathscr{V}$, i.e. α is valid for «*some claim*» if and only if $\mathscr{V}_v(\text{«}\textit{some claim}\text{»}, \alpha) = 1$, where v is the public verification key corresponding to the secret shared signing key.

[4]Note that to enable efficient combination of the shares, a party must know which signature shares are correct. This was not a problem for certificates based on (regular) signatures, since by definition everyone can check their correctness. In the case of *threshold* signatures parties use efficient two-party zero-knowledge protocol for proving the correctness of signature shares (cf. Sect. 2.2.3): when a party P_j provides a signature share σ_j, she also proves to the recipient the correctness of σ_j.

Like the implementation of certificates based on the regular signatures, this new implementation also enables two methods of generating certificates: by *bilateral proofs*, denoted $\alpha := \mathsf{certify}_{\mathsf{zkp}}(\text{«}\ldots\text{»})$, and *protocol-driven*, denoted $\alpha := \mathsf{certify}(\text{«}\ldots\text{»})$. Therefore, the changes in the described protocols are minimal: whenever originally a party was helping to generate a certificate valid for «*some claim*» by providing a signature on «*some claim*», she now provides a *signature share* on «*some claim*» (and proves bilaterally to the recipient the correctness of the share). Also the changes in the security argument are straightforward, so we omit them and proceed to the complexity analysis.

Since a threshold signature has size of only $\mathcal{O}(\kappa)$ bits, replacing the original constructions of $\mathsf{certify}_{\mathsf{zkp}}(\text{«}\ldots\text{»})$ and $\mathsf{certify}(\text{«}\ldots\text{»})$ from the protocol of Sect. 2.3 by the above threshold-signatures based counterparts results in an MPC protocol with total bit complexity $\mathcal{O}(c_I n^2 \kappa + (c_M + c_O) n^3 \kappa)$. That is, we obtain a protocol communicating $\mathcal{O}(n^3\kappa)$ bits per multiplication or output gate, and $\mathcal{O}(n^2\kappa)$ bits per input gate.

Summarizing, we obtain the following theorem.

Theorem 2 *Assuming the existence of homomorphic public-key encryption, digital signatures, and threshold signatures (cf. Sections 2.2.1, 2.2.2, and 2.2.3), there exists a protocol allowing n parties connected by an asynchronous network to securely evaluate any circuit, even in the presence of a computationally bounded adversary actively corrupting up to $t < n/3$ parties.*

The bit complexity of the protocol is $\mathcal{O}(c_I n^2 \kappa + (c_M + c_O) n^3 \kappa)$, where c_I, c_M, c_O denote the number of input, multiplication, and output gates, respectively, and κ is a security parameter. The round complexity of the protocol is linear in the depth of the circuit.

2.5 MPC with quadratic communication

While the protocol presented in the previous section is significantly more efficient than the previous protocols for optimally resilient asynchronous MPC (cf. [BCG93, BKR94, SR00, PSR02]), it falls short of matching the bit complexity of currently most efficient MPC protocols for *synchronous* networks. In particular, Hirt and Nielsen [HN06] have recently proposed a protocol for synchronous MPC, which communicates only $\mathcal{O}(n\kappa)$ bits per multiplication gate (plus $\mathcal{O}(n)$ of broadcasts, independent of the size of the circuit being evaluated).

In this section we present a different protocol for asynchronous MPC, which communicates only $\mathcal{O}(n^2\kappa)$ bits per multiplication gate and thus narrows the gap between synchronous and asynchronous protocols. This new protocol results from a combination of the general approach to MPC based on threshold encryption with a segmentation of the circuit and a circuit-randomization technique of Beaver [Bea91]. The main result of this section is summarized in the following theorem.

Theorem 3 *Assuming the existence of homomorphic public-key encryption, digital signatures, and threshold signatures (cf. Sections 2.2.1, 2.2.2, and 2.2.3), there exists a protocol allowing n parties connected by an asynchronous network to securely evaluate any circuit, even in the presence of a computationally bounded adversary actively corrupting up to $t < n/3$ parties.*

The bit complexity of the protocol is $\mathcal{O}(c_I n^2\kappa + (c_M + c_O)n^2\kappa)$, where c_I, c_M, c_O denote the number of input, multiplication, and output gates, respectively, and κ is a security parameter. The round complexity of the protocol is linear in the depth of the circuit.

2.5.1 Reducing redundancy

The complexity of $\Omega(n^3\kappa)$ bits per multiplication of the protocol from Section 2.4 stems from the high redundancy of the protocol — to achieve robustness we let *every* party play the role of the king (with n slaves helping him), which results in a duplication of substantial amount of work, since essentially the necessary computation is performed n times. Note that with a passive adversary it would be sufficient to have just one king, but with an active adversary we cannot rely on any particular party to correctly play the role of the king.

Below we describe how to use the work of the parties in a more economic way. The idea is to partition the circuit, so that different kings are responsible for different (multiplication) gates — in this way we avoid redundancy in the performed computation. This however has to be done in a robust way, so that adversary controlling some of the parties cannot prevent progress of computation by "blocking" the gates assigned to the corrupted parties.

The key to achieving robustness in an efficient way is the so-called *circuit-randomization technique* due to Beaver [Bea91]. Using this technique a multiplication of two secret values is performed with help of a *pre-generated* random product triple, which in our case is just a triple of

Let A, B and U, V, W be ciphertexts known to all parties, containing plaintexts $a, b, u, v, w \in \mathcal{M}$ respectively, such that u, v, w are random subject to $u \cdot v = w$. To compute a ciphertext C of $c = a \cdot b$, every party P_i proceeds according to the following instructions:

1. compute $X := A \oplus U$ and $Y := B \oplus V$
2. compute decryption shares and corresponding validity proofs:
$$x_i := \mathscr{D}_i(X) \qquad \beta_i := \mathsf{certify}_{\mathsf{zkp}}(\text{«}x_i \textit{ is valid}\text{»})$$
$$y_i := \mathscr{D}_i(Y) \qquad \gamma_i := \mathsf{certify}_{\mathsf{zkp}}(\text{«}y_i \textit{ is valid}\text{»})$$
3. send (x_i, β_i) and (y_i, γ_i) to all parties
4. collect sets $\mathcal{X} := \{(x_j, \beta_j)\}$ and $\mathcal{Y} := \{(y_j, \gamma_j)\}$, each containing t_{D} correct decryption shares, with corresponding validity proofs.
5. compute plaintexts $x := \mathscr{D}(X, \mathcal{X})$ and $y := \mathscr{D}(Y, \mathcal{Y})$.
6. compute $Z := \mathscr{E}(x \cdot y, r_0)$ for some constant r_0, and output
$$C := Z \ominus (x \star V) \ominus (y \star U) \oplus W \ .$$

Figure 2.9: Randomization technique of Beaver [Bea91] for multiplication of encrypted values (code for party P_i).

ciphertexts (U, V, W) containing secret random plaintext values $u, v, w \in \mathcal{M}$, such that $u \cdot v = w$ holds. Given such a triple and two ciphertexts A, B containing plaintexts a, b, we can compute a ciphertext C of $c = a \cdot b$ by *publicly decrypting* $A + U$ and $B + V$, and by using the homomorphic property of the encryption and the following identity

$$\begin{aligned} a \cdot b &= ((a+u) - u) \cdot ((b+v) - v) \\ &= (a+u) \cdot (b+v) - (a+u) \cdot v - u \cdot (b+v) + u \cdot v \\ &= (a+u) \cdot (b+v) - (a+u) \cdot v - u \cdot (b+v) + w \ . \end{aligned}$$

The corresponding protocol is given in Fig. 2.9. It is easy to see that this protocol uses $\mathcal{O}(n^2\kappa)$ bits of communication: essentially, we need two threshold decryptions, where every player sends his decryption share to all players, together with a certificate of correctness, for which we use threshold signatures (cf. Sect. 2.4). Therefore to obtain an overall $\mathcal{O}(n^2\kappa)$

communication per multiplication gate it remains to generate random product triples using $\mathcal{O}(n^2\kappa)$ bits of communication per triple. The design of an appropriate protocol is the main topic of the next section. Given such a protocol, we obtain the desired result, i.e. an asynchronous MPC protocol with $\mathcal{O}(n^2\kappa)$ bits of communication per gate (cf. Theorem 3).

2.5.2 Generating random triples

A *random triple* is a tuple of three ciphertexts (U, V, W) containing secret plaintexts $u, v, w \in \mathcal{M}$, respectively, which are chosen uniformly at random, subject to $u \cdot v = w$. Since we are considering an active adversary, our protocol for generating random triples will provide additionally a certificate for the correctness of each generated triple. This certificate will be a threshold signature on the message «(U, V, W)*: a correct triple* id», where id is a unique identifier assigned according some pre-agreed convention.

Before we describe in detail the proposed protocol for generating random triples, we give an intuitive overview of its structure. First we need a protocol **`one-triple`** allowing a party P_k (playing the role of a king) to generate its own random certified triple $(U^{(k)}, V^{(k)}, W^{(k)}; \sigma^{(k)})$. Furthermore, we need a protocol **`select`**, to which each party P_i provides its own random triple $(U^{(i)}, V^{(i)}, W^{(i)}; \sigma^{(i)})$ as input, and which yields a set of at least $n - t$ valid triples as output, on which all (honest) parties agree. Given these two sub-protocols, the high-level structure of **`gen-triples`** protocol is the following:

1. Every party P_k starts an instance of **`one-triple`**(id, k) protocol to generate a certified random triple $(U^{(k)}, V^{(k)}, W^{(k)}; \sigma^{(k)})$ (all parties P_j play roles of slaves to help P_k)

2. All parties start **`select`**, where party P_i uses as its input the triple $(U^{(i)}, V^{(i)}, W^{(i)}; \sigma^{(i)})$ generated in the previous step. When **`select`** terminates, parties have agreed on a set of at least $n - t$ valid triples $\{(U^{(j)}, V^{(j)}, W^{(j)})\}_{j \in J}$.

Protocol **`select`** was presented already in Section 2.3.5.1, where we used it to select inputs to the computation (Fig. 2.2). The starting point for **`select`** is a scenario where each party P_i has some input value X_i and

To generate for P_k a certified random ciphertext (U, α), with α valid for «U*: 1st part of triple* $\mathsf{id}(k)$» parties proceed as follows:

GENERATION: code for every P_i:

1. pick random $u_i \in \mathcal{M}$ and compute $U_i := \mathscr{E}(u_i)$
2. construct $\beta_i = \mathsf{certify}_{\mathsf{zkp}}$(«$P_i$ *knows* u_i *in* U_i»)
3. compute $\sigma_i := \mathsf{Sign}_i$(«$U_i$: *component of 1st part of triple* id*(k)*»)
4. send (U_i, β_i, σ_i) to P_k:

CONSTRUCTION: code for P_k:

1. collect a set $S_{\mathsf{id}(k)} := \{(U_i, \beta_i, \sigma_i)\}_{i \in I_{\mathsf{id}(k)}}$, $|I_{\mathsf{id}(k)}| \geq t + 1$,
   with each β_i valid for «P_i *knows* u_i *in* U_i»,
   and each σ_i valid for «U_i : *component of 1st part of triple* id*(k)*».
2. send $S_{\mathsf{id}(k)}$ to all parties; each P_i computes $U := \bigoplus_{i \in I_{\mathsf{id}(k)}} U_i$,
   and helps to construct α in the next step.
3. construct $\alpha := \mathsf{certify}$(«$U$*: 1st part of triple* $\mathsf{id}(k)$»).
4. output (U, α).

Figure 2.10: Protocol **`random`**(id, k): The code for generating a certified random value with $\mathsf{id}(k)$ for king P_k .

a corresponding validity certificate β_i. The goal of **`select`** is distribution and agreement on a subset of at least $n - t$ individual inputs. See Figure 2.2 for details.

In our construction of **`one-triple`** we need to generate certified, encrypted random values, so first we present a sub-protocol **`random`** (Fig. 2.10), which achieves exactly that. More precisely, **`random`** allows a party P_k, acting as a king, to generate a ciphertext U with a certificate α, where the plaintext hidden in U is random and unknown to any party, and α certifies this fact. Given (U, α), king P_k can extend it to a random triple using **`one-triple`**, as presented in Fig. 2.11. Note that this protocol works similarly to the protocol for generating a randomizer during the multiplication step (cf. Figs. 2.4 and 2.5). In particular, when comput-

To generate for P_k a certified random ciphertext $(U, V, W; \beta)$, with β valid for «(U, V, W)*: correct triple* $\mathsf{id}(k)$» parties proceed as follows:

REQUEST: code for P_k:

1. run **random**(id, k) to generate a tuple (U, α), with α valid for «U*: 1st part of triple* $\mathsf{id}(k)$», and send (U, α) to all parties.

REPLY: code for every P_i:

1. wait for (U, α) from P_k
2. compute $V_i := \mathscr{E}(v_i)$ and $W_i = \mathscr{R}(v_i \star U)$ for a random $v_i \in \mathcal{M}$
3. construct
   $\beta_i := \mathsf{certify}_{\mathsf{zkp}}$(«$P_i$ *knows* v_i *in* V_i*, and* W_i *is a randomization of* $v_i \star U$»)
4. compute $\sigma_i := \mathsf{Sign}_i$(«$(V_i, W_i)$: *part of triple* $\mathsf{id}(k)$»)
5. send $(V_i, W_i; \beta_i, \sigma_i)$ to P_k

CONSTRUCTION: code for P_k:

1. collect $T_{\mathsf{id}(k)} := \{(V_i, W_i; \beta_i, \sigma_i)\}_{i \in I_{\mathsf{id}(k)}}$, $|I_{\mathsf{id}(k)}| \geq t + 1$, with each β_i valid for «P_i *knows* v_i *in* V_i*, and* W_i *is a randomization of* $v_i \star U$» and each σ_i valid for «(V_i, W_i) : *part of triple* $\mathsf{id}(k)$».
2. send $T_{\mathsf{id}(k)}$ to all parties; each P_i computes $V := \bigoplus_{i \in I_{\mathsf{id}(k)}} V_i$, $W := \bigoplus_{i \in I_{\mathsf{id}(k)}} W_i$, and helps to construct β in the next step.
3. construct $\beta := \mathsf{certify}$(«$(U, V, W)$*: correct triple* $\mathsf{id}(k)$»).
4. output $(U, V, W; \beta)$.

Figure 2.11: Protocol **one-triple**(id, k) for generating a certified random triple (U, V, W) with $\mathsf{id}(k)$ for king P_k.

ing a certificate β_i for the claim

«P_i *knows* v_i *in* V_i*, and* W_i *is a randomization of* $v_i \star U$»

the variables P_i, V_i, W_i, and U are replaced by the actual values they stand for, while v_i stays in the claim as a literal, since it is just a name for the plaintext hidden in the ciphertext V_i.

On the use of Byzantine Agreement. A closer look at **`gen-triples`** protocol reveals one disturbing property. Recall that to implement multiplication of encrypted values via circuit randomization (cf. Fig. 2.9), we need one (agreed upon) random triple per multiplication gate. If we distribute and select the triples in batches of $(n-t)$ as described above, we would need in total $\lceil c_M/(n-t)\rceil$ runs of **`gen-triples`**, which implies that we would need about c_M runs of Byzantine Agreement, since the protocol **`gen-triples`** executes **`select`**, which in turn uses n Byzantine Agreements. This means that the number of required Byzantine Agreements would depend on the size of the circuit. One can avoid this by applying to **`gen-triples`** the same trick as we used in the case of multiple inputs per player (cf. Section 2.3.7): In the first step each P_k runs $\ell > 1$ instances of **`one-triple`**, to generate ℓ certified random triples, where $\ell = \lceil c_M/(n-t)\rceil$. Then in the second step each party P_i uses all ℓ triples as its input to **`select`**. Since protocol **`select`** returns a set of at least $(n-t)$ inputs, we obtain an agreement on c_M random triples with only n Byzantine Agreements, which is independent of the circuit size.

2.6 Computing randomized functions

As mentioned in Section 2.3.1, the MPC protocols presented so far are restricted to the evaluation of *deterministic* circuits only. However, the modifications enabling the evaluation of *randomized* circuits are pretty straightforward: we introduce additional "random gates"and generate random values using an approach as for generating random triples. More precisely, a random gate G is specified as (G, random) and the value $v(G)$ of the gate is equal to a random value generated jointly by the players, as described below. Then at the beginning of the protocol every party generates a batch of certified random values, using the protocol **`random`** (Fig. 2.10), and subsequently parties agree on the random values from at least $n-t$ parties, using the protocol **`select`** (Fig. 2.2). The resulting certified random values are assigned to the random gates in some pre-agreed deterministic way.

If each party P_i enters **`select`** with a batch of $\lceil c_R/(n-t)\rceil$ certified random values, where c_R is the total number of random gates in the circuit, then we obtain agreement on c_R random values with just n Byzantine Agreements, which is independent of the circuit size. The communication cost per one random gate is $\mathcal{O}(n^2\kappa)$.

2.7 Computing functions with private outputs

To simplify the presentation, until now we considered circuits with public outputs only, i.e., every party learns the output(s) of the circuit. In the following, we present an extension that allows for outputs that are delivered only to an authorized party, say P_j.

A standard way of dealing with this problem in the *synchronous* model is the following: For every secret output value party P_j provides an additional secret random input value, which is added to the output-ciphertext (using the homomorphic property of the encryption scheme) to blind the secret output before decryption. Since only P_j knows the random blinding input, the decrypted blinded value gives no information to any party except P_j, who nevertheless can easily unblind the secret output. Unfortunately, this trick doesn't work in the *asynchronous* setting, as we cannot guarantee that the inputs provided by P_j (including the blinding value) are considered by the parties during the computation (cf. Sect. 2.3.5.1).

The intuition of the protocol is that the decryption shares are not sent to the king, but rather directly to P_j. Every decryption share must go along with a proof of validity. This proof must not be interactive with P_j (the parties cannot wait for messages of P_j), and the proof must not be given to other parties (this would violate the privacy of the output protocol). Therefore, we have every slave P_i blind his decryption share c_i with a random value r_i, i.e., $c'_i = c_i + r_i$, encrypt r_i with randomness ρ_i, i.e., $R_i = \mathscr{E}(r_i, \rho_i)$, and prove interactively towards every player P_l knowledge of r_i such that r_i encrypts to R_i and $c'_i - r_i$ is a valid decryption share. Upon accepting the proof, every player P_l hands a signature share for «(c'_i, R_i) *is a good decryption share for slave* P_i» to P_i, who then sends c'_i, r_i, ρ_i to P_j. Given this information from at least $n - t$ players, P_j picks the valid decryption shares and decrypts his private output.

We note that a similar technique has been recently used, but for a different purpose, by Schoenmakers and Tuyls [ST04].

2.8 Providing inputs in asynchronous networks

A fully asynchronous MPC protocol inherently cannot consider the input of every honest party; once $n-t$ inputs are ready, the protocol must start. This is a serious drawback which makes the fully asynchronous model unusable for many real-world applications. We show that with only few rounds of synchronization we can consider the input of *every* honest party participating in the computation.

This model seems very reasonable in the real-world: the parties would wait for other parties to have their input ready, and if not, use other means of communication (email, phone, fax, etc) to synchronize. However, the MPC protocol itself should run asynchronously to comply with the properties of existing networks, namely that the delay of messages is hard to predict. Note that asynchronous protocols can be looked as "best effort" protocols where the progress in the protocol is as fast as possible with the available network, in contrast to synchronous protocols whose progress is limited by the assumed worst-case delay of the network.

The following slight modification of **`select`** protocol (cf. Fig. 2.2), which is the main part of the input-stage protocol, achieves the desired effect using only four synchronous rounds: Every player P_i moves to the last stage (SELECTION) only when either $|C_i| = n$, or the synchronous rounds have elapsed. More precisely, parties generate their certified inputs, and then send them to all parties using the synchronous communication. If during the generation of certified inputs efficient Σ-protocols are used (cf. [CDN01]), this will cost in total five synchronous rounds.

Note that given a few synchronous rounds at disposal one could in principle use a general synchronous *constant-round* MPC protocol, e.g. the one due to Beaver, Micali and Rogaway [BMR90]. However, such a solution would be rather inefficient in practice, since the known constant-round MPC protocols have a relatively high bit complexity.

Chapter 3

Robust combiners of cryptographic primitives

Many cryptographic schemes are based on unproven assumptions about the difficulty of some computational problems. While there exist assumptions whose validity is supported by decades of research (e.g., factoring or discrete logarithm), many new assumptions offering new possibilities are being proposed in the literature, and it is unclear how to decide which assumptions are trustworthy. Therefore, given multiple implementations of some cryptographic primitive, all based on different assumptions, it is often difficult to decide which implementation is the most secure one.

Robust combiners offer a method of coping with such difficulties: they take as input multiple candidate schemes based on various assumptions, and construct a scheme whose security is guaranteed if at least *some* candidates are secure. That is, the resulting scheme is secure as long as sufficiently many of the assumptions underlying the input candidates are valid. This provides tolerance against wrong assumptions since even a breakthrough algorithm for breaking one (or some) of the assumptions doesn't necessarily make the combined scheme insecure.

More formally, a *(k; n)-robust $\mathcal{A}$-combiner* is a construction which takes as input n implementations of a primitive $\mathcal{A}$, and yields an implementation of $\mathcal{A}$ which is guaranteed to be secure as long as at least k input implementations are secure. Robust combiners for some primitives, like one-way functions or pseudorandom generators, are rather simple, while for others, e.g., for oblivious transfer (OT), the construction of combiners seems considerably harder. In particular, in a recent work Harnik *et al.* [HKN+05] show that the so-called *transparent black-box* $(1;2)$-robust OT-combiners are impossible. That is, they prove that $(1;2)$-robust OT-combiners restricted to the *on-line* use of the input candidates, do not exist (cf. Def. 5 in Section 3.1.2). In the same paper the authors propose also a very simple and efficient transparent black-box $(2;3)$-robust OT-combiner.

In general, the candidates input to the combiner do not have to be necessarily implementing the same primitive, and the goal of a combiner may be a construction of a primitive different from the primitives given at the input. That is, a robust combiner can be viewed as a robust *reduction* of the output primitive to the input primitive(s).

Overview. In this chapter we consider robust combiners for private information retrieval (PIR), bit commitment (BC), oblivious transfer (OT) and oblivious linear function evaluation (OLFE). In particular, in Section 3.2 we present a $(1;2)$-robust PIR-combiner, i.e., a combiner which given two implementations of PIR yields an implementation of PIR which is secure if at least one of the input implementations is secure. We also describe various techniques and optimizations based on the properties of existing PIR protocols, which yield PIR-combiners with better efficiency and wider applicability.

In Section 3.3, we construct so-called $\mathcal{A}$-to-$\mathcal{B}$ combiners, i.e. *cross-primitive* combiners, which given multiple implementations of a primitive $\mathcal{A}$ yield an implementation of some other primitive $\mathcal{B}$, where the resulting implementation is guaranteed to be secure assuming that sufficiently many of the input implementations of $\mathcal{A}$ are secure. Specifically, we construct $(1;2)$-robust PIR-to-BC and PIR-to-OT combiners. To the best of our knowledge these are the first combiners of this type. In addition to being interesting in their own right, such combiners offer also insights into relationships and reductions between cryptographic primitives. In particular, our PIR-to-OT combiner together with the impossibility result of [HKN+05] rule out certain types of reductions of PIR to OT. While a reduction of OT to PIR has been demonstrated a while ago [DMO00], no reduction in the opposite direction is known so far. The presented separation gives a partial explanation why this reverse reduction is missing, and implies that a successful reduction would have to make an *off-line* use of the given OT (cf. Definition 5 in Section 3.1.2 and Corollary 5 in Section 3.3.2)

In Section 3.4 we suggest a more fine-grained approach to the design of robust combiners. We argue that in order to obtain combiners as efficient as possible, the constructions may take into account that some properties of the input candidates are proved to hold unconditionally, and hence cannot fail even if some computational assumptions turn out to be wrong. Therefore, keeping in mind the original motivation for combiners, i.e., the protection against wrong assumptions, we propose stronger and more general definitions of robust combiners for two-party primitives, which enable a more fine-grained approach to the design of combiners. In particular, the new definitions capture scenarios where in the candidate implementations the security of one party is based on an assumption different from the assumption underlying the security of the other party, or where the security of one party is unconditional. This finer distinction can then be exploited in constructions of combiners.

For these new definitions we propose OT-combiners yielding secure OT when the total number of candidates' failures on side of either party is strictly smaller than the number of candidates. In particular, we propose an OT-combiner which guarantees secure OT even when only one candidate is secure for both parties, and all the remaining candidates are insecure for one of the parties. Moreover, we propose also an efficient *uniform* OT-combiner, i.e. a single combiner which is secure *simultaneously* for a wide range of candidates' failures. We show optimal robustness of the proposed combiners by proving a *very simple, yet stronger* impossibility result for OT-combiners. Whereas the impossibility proof in [HKN$^+$05] only proofs the non-existence of *transparent black-box* combiners, our proof excludes *all types* of combiners. This is another nice consequence of our stronger definitions. Since our new definition is stronger than the previous definition, all proposed constructions satisfy also the latter, and we obtain tight bounds also for the previous definition.

Finally, we propose combiners for oblivious linear function evaluation (OLFE), a primitive which is a generalization of OT (cf. Sect. 3.1.1). The first presented OLFE-combiner has several advantages over some of the OT-combiners. In particular, it is perfect (its error probability is equal zero), and it uses any candidate instance only once, which is optimal. Additionally, an interesting feature of this combiner is that it makes use of secret sharing techniques inspired by multi-party computation. The second proposed construction uses techniqes used for OT-combiners to yield an *uniform* OLFE-combiner.

The results presented in this chapter are a joint work with Remo Meier and Jürg Wullschleger [MP06, MPW07, PW06].

3.1 Preliminaries

We first review shortly the primitives relevant in this chapter, and for more formal definitions we refer to the literature. Then we recall definitions of robust combiners. The parties participating in the protocols and the adversary are assumed to be probabilistic polynomial time Turing machines (PPTMs).

Notation. If x is a bit-string, $|x|$ denotes its length, and we write $x\|y$ to denote the concatenation of the bit-strings x, y. For an integer $m > 0$ we write $[m]$ to denote the set $\{1, \ldots, m\}$. We use $\mathbb{F}$ to denote an arbitrary finite field, and $\mathbb{F}_q$ to denote the finite field with k elements. We use regular capital letters to denote a primitive, and sans-serif capital letters to denote an instance, i.e., a concrete implementation of a primitive. For example, PIR denotes the primitive of private information retrieval, and PIR_1, PIR_2 denote two instances of PIR.

3.1.1 Primitives

Private Information Retrieval. Private Information Retrieval (PIR) is a protocol between two parties, a server holding an m-bit database $x = (x_1\| \ldots \|x_m)$, and a user holding an index $i \in [m]$. The protocol allows the user to retrieve bit x_i without revealing i to the server, i.e. it protects user's privacy. The original notion of PIR, introduced by Chor *et al.* [CKGS98], makes use of multiple servers holding the database and guarantees information-theoretic security for the user, assuming that after the initial set-up the servers don't communicate with each other. In this work we consider only *single-database* PIR, introduced by Kushilevitz and Ostrovsky [KO97], where there is only one server holding the database and where the security of the user is based on a computational assumption. Of interest are only *non-trivial* protocols, in which the total *server-side* communication (i.e. communication from the server to the user) is less than m bits. Moreover, of special interest are 2-message protocols, in which only two messages are sent: a *query* from the user to the server and a *response* from the server to the user.

Oblivious Transfer. Oblivious Transfer (OT), originally introduced by Rabin [Rab81], is also a two-party primitive, and in the literature many variants of OT have been defined and studied. The variant described here and used in this work is more precisely denoted as *1-out-of-2 bit-OT*, and it was introduced by Even *et al.* [EGL85]. It is a protocol between a sender holding two bits b_0 and b_1, and a receiver holding a choice-bit c. The protocol allows the receiver to obtain the bit b_c so that the sender does not learn any information about receiver's choice c, and the receiver does not learn any information about the bit b_{1-c}. Some other variants of OT include *Rabin's OT*, *1-out-of-m bit-OT*, or *1-out-of-m string-OT*, but all are known to be equivalent [Rab81, Cré87, CK88].

Weak Oblivious Transfer. Weak Oblivious Transfer ((p, q)-WOT) is an oblivious transfer with relaxed privacy guarantees for the participants [DKS99]: with probability at most p a cheating sender learns which bit the receiver has chosen to receive, and with probability q a cheating receiver learns both input bits of the sender.

Oblivious Linear Function Evaluation. Oblivious Linear Function Evaluation (OLFE) over a finite field $\mathbb{F}$ is a natural generalization of oblivious transfer for domains larger than one bit. In OLFE over $\mathbb{F}$ the sender's input is a linear function $f(x) = a_1 x + a_0$, where $a_0, a_1, x \in \mathbb{F}$, and the receiver's input is an argument $c \in \mathbb{F}$. The goal of OLFE is that the receiver learns the value of sender's function at the argument of his choice, i.e. he learns $y = f(c)$ (and nothing else), and the sender learns nothing. To see that oblivious transfer is indeed a special case of OLFE, consider OLFE over $\mathbb{F}_2$: it can be easily verified, that the output bit b_c of oblivious transfer with inputs (b_0, b_1) and c respectively, can be interpreted as the evaluation of a linear function $f(c) = a_1 \cdot c + a_0$ over $\mathbb{F}_2$, since

$$b_c = \underbrace{(b_0 + b_1)}_{\equiv a_1} \cdot c \; + \; \underbrace{b_0}_{\equiv a_0} \; .$$

Bit Commitment. Bit Commitment (BC) [Blu81, Blu82] is a two-phase protocol between two parties Alice and Bob. In the *commit* phase Alice commits to a bit b without revealing it, by sending to Bob a bit-string e, which is an "encrypted" representation of b. Later, in the *decommit* phase, Alice sends to Bob a decommitment string d, allowing Bob to "open" e and obtain b. In addition to the correctness, a bit commitment scheme must satisfy two properties: *hiding*, i.e., Bob does not learn the bit b before the decommit phase, and *binding*, i.e., Alice cannot come up with two decommitment strings d, d' which lead to opening the commitment e as different bits. We consider also *weak* bit commitment, i.e. BC with *weak binding* property: Alice might be able to cheat, but Bob catches her cheating with noticeable probability [BIKM99].

3.1.2 Robust combiners

The following definition is a generalization of the definition of combiners given in [HKN+05], and it can be viewed also as a generalization of a *reduction* of a primitive $\mathcal{B}$ to a primitive $\mathcal{A}$. Note that a $\mathcal{A}$-to-$\mathcal{B}$ combiner can be simply realized by first running an $\mathcal{A}$-combiner, and then constructing an instance of $\mathcal{B}$ from the instance of $\mathcal{A}$ resulting from the combiner. However, this more general approach has some potential advantages. First, it might be more efficient to directly combine instances of $\mathcal{A}$ to an instance of $\mathcal{B}$, instead of going sequentially through an $\mathcal{A}$-combiner and a reduction. Second, it might be possible to combine $\mathcal{A}$-to-$\mathcal{B}$ directly even when no reduction of $\mathcal{B}$ to $\mathcal{A}$ is known or possible (and hence the simple approach cannot be applied).

Definition 2 ((k;n)-robust $\mathcal{A}$-to-$\mathcal{B}$ combiner) *Let $\mathcal{A}$ and $\mathcal{B}$ be cryptographic primitives. A* (k;n)-robust $\mathcal{A}$-to-$\mathcal{B}$ combiner *is a PPTM which gets n candidate schemes implementing $\mathcal{A}$ as inputs, and implements $\mathcal{B}$ while satisfying the following two properties:*

1. *If at least k candidates securely implement $\mathcal{A}$, then the combiner securely implements $\mathcal{B}$.*
2. *The running time of the combiner is polynomial in the security parameter κ, in n, and in the lengths of the inputs to $\mathcal{B}$.*[1]

An $\mathcal{A}$-to-$\mathcal{A}$ combiner *is called an* $\mathcal{A}$-combiner.

For completeness we recall three definitions from [HKN+05], which will be useful in our constructions. These definitions introduce several types of restricted combiners, in particular so-called *black-box* combiners and their variants, imposing limitations on how a combiner can use the candidate implementations and on the methods of proving the security of the combiner. Note that the three notions of black-box combiners are

[1]Here an implicit assumption is made, that the candidates themselves run in polynomial time (cf. Sect. 3.1.3)

equivalent for non-interactive primitives, like for example one-way functions. However, in the case of interactive primitives there are significant differences between them, as we explain below.

Definition 3 (Black-box combiner) *A (1; 2)-robust combiner is called a (fully)* black-box combiner *if the following two conditions hold:*

BLACK-BOX IMPLEMENTATION: *The combiner is an oracle PPTM given access to the candidates via oracle calls to their implementation function.*

BLACK-BOX PROOF: *For every of the two candidates input to the combiner there exists an oracle PPTM $R^{\Box}$, such that if an adversary A breaks the combiner, then R^A breaks the candidate.*[2]

Definition 4 (Third-party black-box combiner) *A* third-party black-box combiner *is a black-box combiner where the input candidates behave like trusted third parties. The candidates give no transcript to the players but rather take their inputs and return outputs.*

Definition 5 (Transparent black-box combiner) *A* transparent black-box combiner *is a black-box combiner for an interactive primitive where every call to a candidate's next message function is followed by this message being sent to the other party.*

There are several reasons for introducing the above distinctions. First note that an unrestricted combiner could totally ignore the input candidates and implement the output primitive directly, from scratch. The notion of *black-box* combiners excludes this possibility and captures combiners yielding implementations whose security relies on the security guarantees of the input candidates. Moreover, we would like to have combiners as simple as possible, and the defined types allow for an objective comparisons between different constructions. Finally, the classification of combiners to the corresponding types partially explains which properties and parameters of the studied primitives are significant in the design of combiners for them.

Third-party black box combiners are most desirable, since they rely only on the functionality and security of the primitives, ignoring all implementation details (including the protocol messages). This is quite a serious limitation, as it is impossible to construct a one-way function from

[2]In the case of a $(k; n)$-robust combiner the corresponding requirement states that at least $n - k + 1$ candidates can be broken in this way. Since we focus on (1; 2)-robust combiners, we omit the details for general k, n.

a third-party implementation of OT [HKN+05]. Transparent black-box combiners allow the use of the transcript of the protocols, but only in an *on-line* fashion, where the protocol messages generated by the oracles implementing the input primitives must be always transferred to the receiver as specified by the input protocol. A (fully) black-box combiner is additionally allowed an unlimited *off-line* access to the oracles implementing the input protocols. This gives the combiner the most power, but might be less desirable in practice. For example, if some two-party primitive is given as two physical devices, one for each party, an off-line access to the primitive would require an off-line access to both devices, which might be hard to realize in practice.

As mentioned previously, an $\mathcal{A}$-to-$\mathcal{B}$ combiner can be viewed as a generalization of a *reduction* of a primitive $\mathcal{B}$ to a primitive $\mathcal{A}$. In other words, a reduction of a primitive $\mathcal{B}$ to a primitive $\mathcal{A}$, i.e. a construction of $\mathcal{B}$ from $\mathcal{A}$, is just a $(1;1)$-robust $\mathcal{A}$-to-$\mathcal{B}$ combiner. Therefore, the above definitions also include notions like a *transparent black-box reduction* or a *third-party black-box reduction*.

3.1.3 Remarks on constructions of combiners

Security vs. functionality. As pointed out by Harnik *et al.* [HKN+05], cryptographic primitives are mainly about security, while functionality issues are often straightforward. For example, a PIR protocol has to satisfy a security property, i.e., the privacy of the user, and functionality properties: efficiency, completeness, and non-triviality. Usually[3] the privacy of the user of a PIR scheme is based on some cryptographic assumption, and the remaining properties hold unconditionally. Moreover, in some cases a possible way of dealing with unknown implementations of the primitives is to test them for the desired functionality. That is, even if the candidate input primitives are given as black-boxes, one can test them before applying a combiner (for a more detailed discussion of these issues see Section 3.1 in [HKN+05]).

For the above reasons we assume that the candidates used as input by the combiners are guaranteed to have the desired functionality. For example, we assume that candidate PIR-schemes satisfy efficiency, completeness, and non-triviality, and that explicit bounds on running time and on communication complexity are given as parts of the input to the

[3]We are not aware of any (single-database) PIR protocol not conforming to this characterization.

combiner. Thus, the task of a combiner is to protect against wrong computational assumptions. This approach is especially relevant in the context of private information retrieval, since some of the most efficient PIR protocols are based on new computational assumptions (e.g., [CMS99, KY01]), which are less studied and so potentially more likely to be broken (cf. recent attack of Bleichenbacher *et al.* [BKY03] on [KY01]).

Efficiency considerations. Recall that the basic definition of robust combiners (Def. 2) is not very demanding in terms of efficiency — it is only required, that the running time of a combiner, hence also its space and communication complexities, are polynomial in the adequate parameters, and no other restrictions are imposed. Obviously, the lower the total complexity the better, but the total complexity of a combiner depends on the complexities of the candidates. For example, the total running time of a combiner can change dramatically when the roles of candidates with significantly different running times are switched.

Therefore, sometimes instead of considering the *total* running time or communication complexity of a (black-box) combiner, we measure the efficiency of a combiner by considering its computation or communication "outside"the candidates and by counting the number (and possibly the size) of calls to the candidates. In other words, it is desirable that a combiner uses the input candidates only few times and on relatively small inputs[4], and we use this criterion as a guideline in constructions of combiners.

On the other hand, it is clear that a black-box $(1;n)$-robust combiner must use every input candidate at least once: if some candidate, say the j-th one, is ignored by combiner, then the combiner will fail on inputs where only the j-th candidate is a secure implementation of the input primitive. This implies that in many cases the best we can hope for is that the time complexity of an implementation resulting from a combiner is proportional to the *sum* of the complexities of the candidates.

3.1.4 Tools

In this section we describe shortly two tools used in our constructions: Shamir's secret sharing scheme via polynomials [Sha79], and two special-purpose combiners for oblivious transfer, offering protection against insecure implementations for one party only [HKN+05].

[4]This condition is important only for primitives with inputs of variable lengths, like PIR.

Shamir's secret sharing. *Secret sharing* [Bla79, Sha79] allows a party to distribute a secret among a group of parties, by providing each party with a *share*, such that only authorized subsets of parties can collectively reconstruct the secret from their shares. We say that a sharing among n parties is a k-out-of-n secret sharing, if any k correct shares are sufficient to reconstruct the secret, but any subset of less than k shares gives no information about the secret. A simple method for k-out-of-n secret sharing was proposed by Shamir [Sha79]: a party P having secret value $s \in \mathbb{F}_q$ where $q > n$, picks a random polynomial $f(x)$ over $\mathbb{F}_q$, such that $f(0) = s$ and the degree of $f(x)$ is (at most) $k-1$. A share for party P_i is then computed as $s_i := f(z_i)$, where $z_1, \ldots, z_n$ are fixed, publicly known, distinct non-zero values from $\mathbb{F}_q$. Since the degree of $f(x)$ is at most $k-1$, any k shares are sufficient to reconstruct $f(x)$ and compute $s = f(0)$ (via Lagrange interpolation). On the other hand, any $k-1$ or fewer shares give no information about s, since they can be consistently completed to yield a sharing of any arbitrary $\overline{s} \in F[q]$, and the number of possible completions is the same for every $\overline{s}$.

Special-purpose OT-combiners. Harnik *et al.* [HKN$^+$05] proposed two special-purpose combiners for oblivious transfer, $\mathbf{R}$ and $\mathbf{S}$, which aim at protecting the security of one party only. These combiners are based on techniques of Crépeau and Kilian [CK88], and were used as building blocks in a construction of a (2; 3)-robust OT-combiner. For completeness, we recall these combiners below, in a slightly generalized version.

The combiner $\mathbf{R}$ takes as input n candidates for OT, and combines them into an OT protocol, while maintaining the security of the *receiver*. That is, the resulting OT protocol is guaranteed to be secure for the receiver as long as at least one of the candidates is secure for the receiver. If any candidate is *insecure* for the sender, the resulting OT is *insecure* for the sender. Given the inputs (b_0, b_1) for the sender and c for the receiver, the combiner $\mathbf{R}$ works as follows:

$\mathbf{R}(\mathsf{OT}_1, \ldots, \mathsf{OT}_n)(b_0, b_1; c)$:

1. The sender picks random bits $r_1^0, r_2^0, \ldots, r_n^0$, s.t. $b_0 = r_1^0 \oplus r_2^0 \oplus \cdots \oplus r_n^0$, and sets $r_i^1 := r_i^0 \oplus b_0 \oplus b_1$, for every $i = 1 \ldots n$.
2. The receiver picks random bits $c_1, c_2, \ldots, c_n$, s.t. $c = c_1 \oplus c_2 \oplus \cdots \oplus c_n$.
3. For every $i = 1 \ldots n$ parties run $\mathsf{OT}_i(r_i^0, r_i^1; c_i)$. From i-th execution the receiver obtains output $r_i^{c_i}$.

4. The receiver outputs b_c computed as the XOR of his outputs from all executions, i.e.
$$b_c = r_1^{c_1} \oplus r_2^{c_2} \oplus \cdots \oplus r_n^{c_n} \ .$$

It is easy to verify the correctness of the above construction. For the security of the receiver observe that his choice bit c is encoded as XOR of n random bits, $c_1 \oplus c_2 \oplus \cdots \oplus c_n$, so as long as at least one OT_i is secure for the receiver, the value of c remains unknown to the sender. Note also, that the combiner preserves the security of the sender — when all n candidates are secure for the sender, so is the resulting OT protocol.

The combiner $\mathbf{S}$ takes as input n candidates for OT and combines them into an OT protocol, while maintaining the security of the *sender*. That is, the resulting OT protocol is guaranteed to be secure for the sender as long as at least one of the candidates is secure for the sender. If any candidate is *insecure* for the receiver, the resulting OT is *insecure* for the receiver. Given the inputs (b_0, b_1) for the sender and c for the receiver, the combiner $\mathbf{S}$ works as follows:

$\mathbf{S}(\mathsf{OT}_1, \ldots, \mathsf{OT}_n)(b_0, b_1; c)$:

1. The sender picks random bits $r_1^0, r_2^0, \ldots, r_n^0$, and $r_1^1, r_2^1, \ldots, r_n^1$, such that
$$b_0 = r_1^0 \oplus r_2^0 \oplus \cdots \oplus r_n^0 \quad \text{and} \quad b_1 = r_1^1 \oplus r_2^1 \oplus \cdots \oplus r_n^1 \ .$$
2. For every $i = 1 \ldots n$ parties run $\mathsf{OT}_i(r_i^0, r_i^1; c)$. From i-th execution the receiver obtains output r_i^c.
3. The receiver outputs b_c computed as the XOR of his outputs from all executions, i.e.
$$b_c = r_1^c \oplus r_2^c \oplus \cdots \oplus r_n^c \ .$$

As previously, it is straightforward to verify the correctness of this construction. The security of the sender follows by an analogous argument, since every input bit b_i is encoded as XOR of n random bits, $r_1^i \oplus r_2^i \oplus \cdots \oplus r_n^i$. Therefore, if at least one OT_i is secure for the sender, the value of at least one b_i remains unknown to the receiver. Finally note that when all n candidates are secure for the receiver, so is the resulting OT protocol.

3.2 Combiners for private information retrieval

In this section we assume that two (non-trivial) private information retrieval schemes are given, PIR_1 and PIR_2, where PIR_1 is a *two-message* PIR protocol with a query $q = Q_1(i)$ and a response $r = R_1(q, (x_1\| \dots \|x_m))$, and where PIR_2 is an arbitrary (possibly multi-round) PIR protocol. We use $c_{s,1}(m)$ and $c_{s,2}(m)$ to denote the server-side communication complexities of the PIR-schemes, and $c_{u,1}(m)$ and $c_{u,2}(m)$ to denote the corresponding user-side complexities[5]. Without loss of generality we assume that these complexities give the exact number of communicated bits and are not just upper bounds.

First we describe a basic scheme for a (1;2)-robust PIR-combiner, and then present some variations of the scheme, resulting in better efficiency. Our constructions are black-box combiners, but not *transparent* black-box combiners because they require an off-line access to one of the candidates.

3.2.1 The basic scheme

Our basic PIR-combiner works as follows: to enable the retrieval of the i-th bit from a database $x = (x_1\| \dots \|x_m)$, the server first defines m auxiliary databases $y^1, ..., y^m$, where y^j is just a copy of x rotated by $(j-1)$ positions, i.e.

$$y^j = (x_j\| \dots \|x_{m-1}\|x_m\|x_1\| \dots \|x_{j-1}).$$

The user picks a random $t \in [m]$ and sends to the server PIR_1-query $q = Q_1(t)$. For each database y^j, $j \in [m]$, the server computes the corresponding response $r_j = R_1(q, y^j)$, but instead of sending the responses back to the user, he stores them in a new database $x' = (r_1\| \dots \|r_m)$.[6] Note that the new database x' contains a PIR_1-response for each bit x_j of the original database x, but with the positions rotated by $(t-1)$. Finally the user retrieves bit-by-bit the response r_k for $k = ((i-t) \mod m) + 1$, by running $c_{s,1}(m)$ instances of PIR_2, and computes x_i from r_k. Figure 3.1 presents the combiner in more detail, and the following theorem summarizes the properties of the combiner.

[5]In particular, $c_{s,1}(m) = |R_1(q, (x_1\| \dots \|x_m))|$ and $c_{u,1} = |Q_1(i)|$

[6] A similar technique was used in [DIO01] for universal service-providers for PIR.

SERVER'S INPUT: m-bit string $x = (x_1 \| \ldots \| x_m)$

USER'S INPUT: $i \in [m]$

INPUT PIR PROTOCOLS:
PIR$_1$: 2-message, with query $Q_1(j)$ and response $R_1(Q_1(j), x)$
PIR$_2$: arbitrary

Phase I:

1. server defines m databases, y^j for each $j \in [m]$, where
$$y^j = (x_j \| \ldots \| x_{m-1} \| x_m \| x_1 \| \ldots \| x_{j-1})$$
2. user sends to server a PIR$_1$-query $q = Q_1(t)$, for a random $t \in [m]$
3. server computes m PIR$_1$-responses $r_j = R_1(q, y^j)$ for each $j \in [m]$

Phase II:

1. user computes $k = ((i - t) \mod m) + 1$
2. user retrieves from server response r_k bit-by-bit, by running $|r_k|$ instances of PIR$_2$ with $x' = (r_1 \| r_2 \| \ldots \| r_m)$ as server's input
3. user computes x_i from the PIR$_1$-response r_k

Figure 3.1: The basic (1; 2)-robust PIR-combiner.

Theorem 4 *There exists a black-box (1; 2)-robust PIR-combiner for input candidates* PIR$_1$ *and* PIR$_2$*, where* PIR$_1$ *is a 2-message protocol, and where the candidates' server-side communication complexities satisfy*

$$c_{s,1}(m) \cdot c_{s,2}\left(m \cdot c_{s,1}(m)\right) < m \ . \tag{3.1}$$

The user-side communication of the resulting PIR scheme equals

$$c_{u,1}(m) + c_{s,1}(m) \cdot c_{u,2}\left(m \cdot c_{s,1}(m)\right) \ .$$

Proof. Consider the construction presented in Figure 3.1. It is clear, that this an efficient construction with a black-box implementation. Let $\overline{\text{PIR}}$ denote the resulting PIR. We have to argue correctness, security, and non-triviality.

The correctness is given by the correctness of the candidates PIR$_1$, PIR$_2$, and the fact that the response string r_k retrieved in the second phase corresponds to y^k_t, i.e. the t-th bit of y^k. Since

$$y^k = (x_k \| x_{k+1} \| x_{k+2} \| \dots \| x_m \| x_1 \| \dots \| x_{k-1}),$$

it holds that

$$y^k_t = x_{[(k+t-2) \bmod m]+1} = x_{[(((i-t) \bmod m)+1+t-2) \bmod m]+1} = x_i \,.$$

To show that the construction is a (1; 2)-robust combiner, we have to argue that it yields a secure PIR whenever at least one candidate is secure. We first give an intuitive explanation why it is the case, and then we formally prove it.

Since private information retrieval protects only the privacy of the user, only dishonest servers have to be considered. Given that one candidate is secure, the privacy is guaranteed because a dishonest server learns at most either the index t or k, but never both. Having only one index yields no information about i. More precisely, if PIR_1 is insecure, server learns the random index t. However, as in this case PIR_2 remains secure, the server obtains no information about index k when the user retrieves the response r_k, and so does not gain information about the index i. On the other hand, if PIR_2 is insecure, then the server learns index k, but as PIR_1 is now secure, the server gets no information about t. Since t is randomly chosen, the knowledge of k also does not give any information about the index i.

To prove the security more formally (cf. Def. 3), for every of the two input candidates PIR_1, PIR_2 we present a corresponding oracle PPTMs $R_1^{\square}$ resp. $R_2^{\square}$, such that if an adversary A breaks the combiner, then R_1^A breaks PIR_1, and R_2^A breaks PIR_2. We describe only R_1^A explicitely, since R_2^A can be constructed analogously.

Roughly speaking, to break PIR_1 the algorithm R_1^A simulates to A an execution of the combiner with input candidates PIR_1, PIR_2, and during the simulation R_1^A executes PIR_2 on a randomly selected index k. When the adversary A returns a prediction about the index i requested in the simulated combiner, R_1^A uses i and k to compute the unknown index t requested in the execution of PIR_1.

More precisely, when R_1^A receives a PIR_1-query $q_1 = Q_1(t)$ for some unknown index t, it forwards q to A as the message of a user in the first phase of the combiner. Then in the second phase R_1^A uses PIR_2 to retrieves r_k for a random $k \in [m]$. After the simulation of the combiner is completed, the adversary A returns its prediction about the index i requested in the combiner. Given i, k, and the identity $k = ((i - t) \mod m) + 1$, the attacker R_1^A computes and returns t. Clearly, R_1^A is successful in breaking PIR_1 whenever A is successful in breaking the combiner. As mentioned above, R_2^A breaking PIR_2 is constructed analogously.

Finally, we argue the non-triviality condition: it is easy to verify that the server-side communication of $\overline{\mathsf{PIR}}$ is

$$c_s(m) = c_{s,1}(m) \cdot c_{s,2}\left(m \cdot c_{s,1}(m)\right) \ ,$$

and the user-side communication is

$$c_u(m) = c_{u,1}(m) + c_{s,1}(m) \cdot c_{u,2}\left(m \cdot c_{s,1}(m)\right) \ .$$

Thus if $c_s(m) < m$ holds, i.e. if Condition (3.1) is satisfied, then $\overline{\mathsf{PIR}}$ is non-trivial. □

3.2.2 PIR-combiners with lower communication

The basic combiner presented in the previous section is conceptually simple and works well for a wide range of candidate PIR-protocols, but leaves some space for improvements. In this section we describe some variations and optimizations of this basic combiner, which yield significant improvements in the communication efficiency of the resulting PIR-schemes. This results in combiners applicable to a wider range of input candidates.

First we describe how to reduce the cost of querying x' by using several databases in parallel. Then we discuss possible improvements in situations when the candidates support the retrieval of entire blocks of bits, instead of single bits.

Reducing overall communication. In the second phase of the basic PIR-combiner the user retrieves r_k bit-by-bit by running $|r_k| = c_{s,1}(m)$ instances of PIR_2 with server's input x' of length $m \cdot c_{s,1}(m)$. An alternative way of retrieving r_k is the following: we arrange all responses $r_1, \ldots, r_n$ into $l = |r_k|$ databases $x'_1, \ldots, x'_l$, each of length m, where x'_j contains the j-th bits of all responses $r_1, \ldots, r_m$. Then the user obtains r_k by retrieving the k-th bits from the databases $x'_1, \ldots, x'_l$. That is, the user and the server run $|r_k|$ instances of PIR_2, where in the j-th instance server's input is x'_j and user's input k. Thus we obtain the following corollary.

Corollary 1 *There exists a black-box (1; 2)-robust PIR-combiner for input candidates* PIR_1 *and* PIR_2*, where* PIR_1 *is a 2-message protocol, and where the candidates' server-side communication complexities satisfy*

$$c_{s,1}(m) \cdot c_{s,2}(m) < m \ .$$

The user-side communication of the resulting PIR scheme equals

$$c_{u,1}(m) + c_{s,1}(m) \cdot c_{u,2}(m) \ .$$

Note that if PIR_2 is also a 2-message PIR protocol, then only *one query* must be sent in the second phase of the combiner (for which $c_{s,1}(m)$ PIR_2-responses will be sent), thus reducing the user-side communication of the resulting PIR scheme even further, to merely $c_{u,1}(m)+c_{u,2}(m)$. Moreover, the resulting PIR is also a 2-message protocol, since the user can send his PIR_2-query already in the first phase, together with the PIR_1-query, and the server can send all the PIR_2-responses as one message. The following corollary summarizes the above observations.

Corollary 2 *There exists a black-box (1; 2)-robust PIR-combiner for 2-message input candidates* PIR_1 *and* PIR_2*, where the candidates' server-side communication complexities satisfy*

$$c_{s,1}(m) \cdot c_{s,2}(m) < m \ .$$

The resulting PIR scheme is also a 2-message protocol, and its user-side communication equals

$$c_{u,1}(m) + c_{u,2}(m) \ .$$

Further optimizations and variations. If PIR_2 retrieves entire blocks rather than single bits (for example, the basic PIR protocol of [KO97] does exactly that), than the retrieval of r_k can be substantially sped-up, as it can proceed block-by-block rather than bit-by-bit. Moreover, if $|r_k|$ is not larger than the size of blocks retrieved by PIR_2, than just one execution of PIR_2 is sufficient.

Corollary 3 *There exists a black-box (1; 2)-robust PIR-combiner for input candidates* PIR_1 *and* PIR_2*, where* PIR_1 *is a 2-message protocol,* PIR_2 *retrieves blocks of size at least* $c_{s,1}(m)$*, and where the candidates' server-side communication complexities satisfy*

$$c_{s,2}(m \cdot c_{s,1}(m)) < m \ .$$

The user-side communication of the resulting PIR scheme equals

$$c_{u,1}(m) + c_{u,2}(m \cdot c_{s,1}(m)) \ .$$

Another simple optimization is possible when PIR_1 supports block-wise retrieval, i.e., when each PIR_1-response r_j allows retrieval of ℓ-bit blocks. In such a case it is sufficient to store in x' a subset of $\lceil m/\ell \rceil$ responses, so that the corresponding blocks cover the entire database — then in Phase II user simply retrieves the block containing the desired bit x_i.

Finally, when the user-side communication of the candidate PIRs is much higher than the server-side communication, it is possible to balance the load better between the two parties with the so called *balancing technique*, which was introduced in the context of information-theoretic PIR [CKGS98], and which can be viewed as a simulation of block-wise retrieval: server partitions the database to u databases of size $\lceil m/u \rceil$. User provides then a single query for some index $j \in [\lceil m/u \rceil]$, which is answered for each of u databases, yielding a block of u bits.

Clearly, one can use multiple optimizations together (if applicable) to obtain the most efficient construction for the given candidate PIR protocols.

3.3 Cross-primitive combiners

The constructions of robust combiners presented so far focused mainly on combining candidate instances of a given primitive $\mathcal{A}$ to yield a secure instance of $\mathcal{A}$. In this section we describe robust combiners of a more general type, combining instances of primitive $\mathcal{A}$ to an instance of primitive $\mathcal{B}$. These *cross-primitive* combiners can be viewed as a combination of robust combiners and reductions between primitives in one construction.

As mentioned in Section 3.1, a $\mathcal{A}$-to-$\mathcal{B}$ combiner can be simply realized by first running an $\mathcal{A}$-combiner, and then constructing an instance of $\mathcal{B}$ from the instance of $\mathcal{A}$ resulting from the combiner. However, this simple approach works only if both the $\mathcal{A}$-combiner and the reduction of $\mathcal{B}$ to $\mathcal{A}$ are known. Thus a main reason to study cross-primitive combiners is that it might be possible to construct an $\mathcal{A}$-to-$\mathcal{B}$ combiner even when no reduction of $\mathcal{B}$ to $\mathcal{A}$ is known or possible (indeed, our PIR-to-OT combiner, presented in Sect. 3.3.2, is such an example). Moreover, a direct $\mathcal{A}$-to-$\mathcal{B}$ combiner might be also more efficient than the application of an $\mathcal{A}$-combiner followed by a reduction.

First we consider the problem of combining PIR protocols to obtain a bit commitment scheme, and present a third-party black-box (1; 2)-robust

PIR-to-BC combiner. Then we turn to the problem of combining PIR to oblivious transfer, and present a black-box (1; 2)-robust PIR-to-OT combiner. The existence of such a combiner is somewhat surprising, given the impossibility result of [HKN+05] and the fact that PIR and OT are very closely related. In fact, this combiner leads to a better understanding of the relation between these two primitives (cf. Corollary 5).

3.3.1 PIR-to-BC combiner

It is well-known that single-database private information retrieval implies one-way functions (OWFs) [BIKM99], which in turn are sufficient to construct computationally hiding and statistically binding bit commitments schemes [Nao91]. It follows immediately that there exists a generic combiner going through these reductions and an OWF-combiner. However, such a combiner is quite inefficient, and it is not a third-party black-box combiner.

In this section we present a more efficient, third-party black-box PIR-to-BC combiner, which is basically a slight variation of the reduction of bit commitment to private information retrieval due to Beimel *et al.* [BIKM99]. In contrast to the generic combiner described above, the BC-scheme resulting from the proposed combiner is statistically hiding and computationally binding. We describe only a construction for *weak* bit commitment, which can then be strengthened by using multiple independent commitments to the same bit [BIKM99]. A detailed description of the combiner is presented in Figure 3.2.

Theorem 5 *There exists a third-party black-box (1; 2)-robust PIR-to-BC combiner yielding a statistically hiding BC, for input candidates* PIR_1 *and* PIR_2 *with server-side communication complexities satisfying*

$$c_{s,1}(n) + c_{s,2}(n) \leq n/2 \; . \tag{3.2}$$

Proof. As mentioned above, it is sufficient to show a combiner from PIR to *weak* bit commitment. Consider the construction presented in Fig. 3.2. It is clear, that the construction is efficient, and it is easy to verify the completeness of the scheme: if both parties play honestly, then Bob accepts, since

$$c \oplus \mathrm{IP}(x, y) = b \oplus \mathrm{IP}(x, y) \oplus \mathrm{IP}(x, y) = b \; .$$

The hiding property is satisfied because the candidate PIR protocols do not transmit enough bits during the commit phase to allow Bob to

ALICE'S INPUT: bit b

BOB'S INPUT: (none)

INPUT PIR PROTOCOLS: PIR_1, PIR_2

commit phase:

1. Alice picks two independent, unif. random strings $x, y \in_R \{0,1\}^\kappa$
   Bob picks two independent, unif. random indices $i_1, i_2 \in_R [\kappa]$, where κ is a security parameter.
2. Alice and Bob execute PIR protocols $\mathsf{PIR}_1(x; i_1)$ and $\mathsf{PIR}_2(x; i_2)$, with Alice as server with database x, and Bob retrieving bit with index i_1 resp. i_2.
3. Alice sends to Bob y and $c = b \oplus \mathrm{IP}(x, y)$
   (where IP denotes inner product over $\mathbb{F}_2$)

decommit phase:

1. Alice sends to Bob x as decommitment string
2. Bob verifies that x is consistent with the bits retrieved during the commit phase, and if verification is successful, he accepts and outputs $b = c \oplus \mathrm{IP}(x, y)$, otherwise he rejects.

Figure 3.2: A (1; 2)-robust PIR-to-(weak)BC combiner.

reconstruct b. More precisely, y is chosen uniform at random and not used in any way by the PIR protocols. It is therefore not possible for a cheating Bob to obtain $\mathrm{IP}(x, y)$ in the second step of the commit phase. After Bob learns y in the third step, he has too little information about x to be able to compute $\mathrm{IP}(x, y)$ with a noticeable advantage. This follows from the assumed bounds on the (server-side) communication complexity of the PIR-protocols and from the lower bounds on the communication complexity of inner product, proved by Chor and Goldreich [CG88] (see [BIKM99] for details).

The weak binding property of the resulting commitment scheme follows from the privacy of PIR candidates. Since it is a (1; 2)-robust combiner, we can assume that at least one candidate is secure. If Alice wants to cheat, she has to change at least one bit of x before sending it to Bob in the decommit phase. However, she does not know at least one of the indices i_1, i_2, as the privacy of one PIR candidate is guaranteed. Bob will therefore catch a cheating Alice with probability at least $1/\kappa$. A more formal proof proceeds similarly to the proof of Theorem 4. For each of the input candidates PIR_1, PIR_2 we construct a corresponding oracle PPTMs $R_1^{\square}$ resp. $R_2^{\square}$, such that if an adversary A breaks the weak binding property of the resulting bit commitment, then R_1^A breaks PIR_1, and R_2^A breaks PIR_2. Roughly speaking, to break PIR_1 the algorithm R_1^A simulates with A an execution of the combiner with input candidates PIR_1, PIR_2, where R_1^A executes PIR_2 on a randomly selected index i_2. By assumption, after the commitment phase the adversary A can cheat by comming up with two different decommitment strings x and x' with probability significantly higher than $1/\kappa$. Thus, R_1^A learns that the bits in x' that differ from the corresponding bits in x are less likely queried by the user in PIR_1 than the bits that are equal. This information clearly compromises the privacy of PIR_1. The machine R_2^A can be constructed analogously.

Summarizing, the construction in Figure 3.2 is a (1; 2)-robust PIR-to-BC combiner, since it is efficient and satisfies completeness, the hiding property, and the weak binding property. Moreover, it is a third-party black-box combiner, since it relies only on the functionality of the input PIR protocols and since any adversary breaking the security of the combiner can be used in a black-box way to break the PIR candidates. $\square$

Obviously, the bound $n/2$ in (3.2) is not tight. Since the focus in this work is on existence of efficient combiners, and since many practical PIR protocols have polylogarithmic communication bounds (which clearly satisfy (3.2)), we do not attempt to optimize this bound. Moreover, the PIR-to-OT combiner presented in the next section implies an alternative, efficient (1; 2)-robust PIR-to-BC combiner (cf. Corollary 4).

3.3.2 PIR-to-OT combiner

The PIR-to-BC combiner presented in the previous section can be viewed as a variation of the general approach to construct (1; 2)-robust PIR-to-BC combiners: first use a construction of an unconditionally hiding BC from a single-database PIR to obtain BC_1 resp. BC_2, and then combine the two BC protocols using the fact that both are unconditionally secure for Alice and at most one not binding (if the corresponding PIR protocol is insecure). As we show in this section, a similar approach works for PIR-to-OT combiners.[7] That is, our proposed PIR-to-OT combiner first con-

[7]Note that unlike in the case of PIR-to-BC combiners, it is unclear whether there exists (1; 2)-robust PIR-to-OT combiner based on combiners for one-way functions: while it is

structs OT protocols OT_1 and OT_2 based on candidates PIR_1 resp. PIR_2, and then combines OT_1 and OT_2 using the fact that both these protocols are unconditionally secure for the sender.

For completeness, Figure 3.3 presents the construction of OT (unconditionally secure for the sender) based on single-database PIR [DMO00]. Intuitively, this protocol is secure for the receiver because PIR protects the privacy of his inputs in Step 1. Therefore the sender cannot tell which indices were used in PIR protocols in Step 1, and which were picked in Step 2. The privacy of the sender is unconditional, since it follows from the non-triviality of PIR — the receiver obtains too little information about the random input strings $x^1, \ldots, x^\eta$ to be able to decode both b_0 and b_1. Note however that in the described protocol the privacy of the sender holds only against *honest-but-curious* receiver. It can however be transformed into a protocol resilient against arbitrary (possibly dishonest) parties [DMO00].

Using the construction from Fig. 3.3, our proposed (1; 2)-robust PIR-to-OT combiner works as follows: given two PIR protocols, PIR_1 and PIR_2, we use this construction to obtain OT protocols OT_1 and OT_2, respectively. Now, as both resulting OTs are unconditionally secure for the sender, we can combine them by using a combiner which guarantees the privacy of the receiver as long as at least one of the two input OTs is secure. For this purpose we use the combiner $\mathbf{R}$ from [HKN$^+$05] (cf. Sect. 3.1.4). Figure 3.4 presents the proposed PIR-to-OT combiner in full detail, and we obtain the following theorem.

Theorem 6 *There exists a black-box (1; 2)-robust PIR-to-OT combiner. In the case of honest-but-curious parties it is a* third-party *black-box combiner.*

Proof. Consider the construction presented in Figure 3.4, and let $\overline{\mathsf{OT}}$ denote the resulting OT scheme. Recall that the construction of OT from PIR ([DMO00], cf. Fig. 3.3) yields an OT protocol with unconditional security for the sender from any non-trivial single-database PIR. Therefore OT_1 and OT_2 are well defined, and it is easy to verify the correctness, efficiency and completeness of $\overline{\mathsf{OT}}$. Moreover, since OT_1 and OT_2 are unconditionally secure for the sender, the combiner R guarantees that also $\overline{\mathsf{OT}}$ is unconditionally secure for the sender. The security of the receiver

known that non-trivial PIR implies one-way functions [BIKM99], it is unlikely that OT can be constructed from one-way functions only [IR89] (the most general assumptions known to be sufficient for OT are the existence of *enhanced* [EGL85, Gol04] or *dense* [Hai04] one-way trapdoor permutations).

SENDER'S INPUT: two bits b_0, b_1

RECEIVER'S INPUT: choice bit c

COMMON INPUTS:
PIR protocol, security param. κ, a param. η polynomial in κ

1. Sender and Receiver execute η runs of PIR,
   with Sender as PIR-server and Receiver as PIR-user:

   for each run $j = 1 \ldots \eta$ they pick independent, unif. random inputs:
   Sender a string $x^j \in_R \{0,1\}^\kappa$, Receiver an index $i^j \in_R [\kappa]$.

   (after this step Receiver knows bits $x^1(i^1), \ldots, x^\eta(i^\eta)$,
   where $x^j(i)$ denotes the i-th bit of x^j).

2. Receiver sets $(i_c^1, ..., i_c^\eta) := (i^1, ..., i^\eta)$,
   and picks random $(i_{1-c}^1, ..., i_{1-c}^\eta) \in_R [\kappa]^\eta$

3. Receiver sends $(i_0^1, ..., i_0^\eta)$ and $(i_1^1, ..., i_1^\eta)$ to Sender.

4. Sender computes

$$\begin{aligned} z_0 &:= b_0 \oplus x^1(i_0^1) \oplus ... \oplus x^\eta(i_0^\eta) \\ z_1 &:= b_1 \oplus x^1(i_1^1) \oplus ... \oplus x^\eta(i_1^\eta) \end{aligned}$$

   and sends z_0, z_1 to Receiver.

5. Receiver computes his output $b_c := z_c \oplus x^1(i^1) \oplus ... \oplus x^\eta(i^\eta)$.

Figure 3.3: Construction of (honest receiver) OT from single-database PIR [DMO00].

in $\overline{\mathsf{OT}}$ follows from the assumption that at least one of the input PIR protocols is secure and from the robustness of R.

Note that for honest-but-curious parties the construction in Fig. 3.3 uses only the inputs and outputs of the underlying PIR scheme. The combiner is therefore *third-party* black-box for honest-but-curious parties. On the other hand, for a dishonest receiver, the construction has to be strengthened by standard techniques based on commitments schemes and zero-knowledge proofs for NP-complete languages. Since any non-trivial PIR implies one-way functions [BIKM99] and one-way functions

SENDER'S INPUT: two bits b_0, b_1

RECEIVER'S INPUT: choice bit c

INPUT PIR PROTOCOLS: PIR_1, PIR_2

1. parties construct OT protocols OT_1, OT_2 from PIR_1 resp., PIR_2 (using the construction from [DMO00]).
2. parties use the special-purpose OT-combiner $\mathbf{R}$ with input candidates OT_1, OT_2, i.e., they run $\mathbf{R}(\mathsf{OT}_1,\mathsf{OT}_2)(b_0, b_1; c)$.

Figure 3.4: A (1; 2)-robust PIR-to-OT combiner.

imply bit commitment [Nao91], the required bit commitments can be constructed from the given PIR-candidates. The known reductions access the transcript of the candidates and consequently cannot be used by third-party combiners. □

Since the OT protocol resulting from the above combiner is unconditionally secure for the sender, we can use it to construct a statistically hiding BC scheme, hence we get the following corollary.

Corollary 4 *There exists a black-box (1; 2)-robust PIR-to-BC combiner yielding a statistically hiding BC.*

Furthermore, recall that a reduction of a primitive $\mathcal{B}$ to a primitive $\mathcal{A}$ can be viewed as a (1; 1)-robust $\mathcal{A}$-to-$\mathcal{B}$ combiner, hence a notion of *transparent black-box reduction* is well-defined. Since in the case of honest-but-curious parties the proposed PIR-to-OT combiner is even a *third-party* black-box combiner, a combination of Theorem 6 with the impossibility result for (1; 2)-robust transparent black-box OT-combiners [HKN+05] leads to the following corollary, which rules out certain types of reductions of PIR to OT. To the best of our knowledge, this is the first result showing a separation between PIR and OT. In particular, the corollary implies that a black-box construction of PIR out of OT working in an on-line way is impossible, i.e., any potential black-box reduction of PIR to OT would have to make off-line use of the given OT.

Corollary 5 *There exists no transparent black-box reduction of single-database private information retrieval to oblivious transfer, even for honest-but-curious parties.*

3.4 Robuster combiners for oblivious transfer

In this section we present new, stronger definitions of robust combiners of two-party primitives, which allow for a more fine-grained approach to the design of combiners. These definitions are motivated by the fact that in many implementations of cryptographic primitives various security properties are based on different, often independent, computational assumptions, or even hold unconditionally, and cannot be broken. In particular, we propose combiners for oblivious transfer and for oblivious linear function evaluation. In addition to leading to combiners which are strictly stronger than the constructions known before, the new definitions allow also for easier impossibility proofs.

3.4.1 Robuster combiners for two-party primitives

If the primitive for which one wishes to construct a combiner is a two-party primitive between Alice and Bob (like for example oblivious transfer or bit commitment), we can make a finer characterization of the security required from the candidates. That is, we can distinguish cases when in the candidate implementations the security of Alice is based on an assumption different from the assumption underlying Bob's security, or when the security of one party is unconditional. For such candidates breaking one assumption doesn't necessarily imply total loss of security (for both parties) and this property can be exploited for the construction of combiners.

Definition 6 (($\alpha, \beta; n$)-robust $\mathcal{A}$-combiner) *Let $\mathcal{A}$ be a cryptographic primitive for two parties Alice and Bob. A* $(\alpha, \beta; n)$-robust $\mathcal{A}$-combiner *is a PPTM which gets n candidate schemes implementing $\mathcal{A}$ as inputs and implements $\mathcal{A}$ while satisfying the following two properties:*

1. *If at least α candidates implement $\mathcal{A}$ securely for Alice, and at least β candidates implement $\mathcal{A}$ securely for Bob, then the combiner securely implements $\mathcal{A}$.*

2. *The running time of the combiner is polynomial in the security parameter κ, in n, and in the lengths of the inputs to $\mathcal{A}$.*[8]

[8]As in Def. 2, we make an implicit assumption that the candidates themselves run in polynomial time (cf. Sect. 3.1.3)

Note that a $(k; n)$-robust combiner is a special case of a $(k, k; n)$-robust combiner, but they are not equivalent. For example, a $(2, 2; 3)$-robust combiner tolerates input candidates $\mathsf{C}_1, \mathsf{C}_2, \mathsf{C}_3$, where the candidate C_1 is secure for Alice only, C_2 is secure for Bob only, and C_3 is secure for both parties, while a $(2; 3)$-robust combiner can fail for such candidates. In other words, the notion of a $(\alpha, \beta; n)$-robust combiner is strictly stronger then that of $(k; n)$-robust combiner, and hence provides better security guarantees.

Another difference between $(k; n)$-robust and $(\alpha, \beta; n)$-robust combiners is that for the new definition it is possible to have "non-uniform" constructions with explicit dependence on α, β. This motivates an even stronger notion of *uniform* combiners. For example, even if there exists a $(\alpha, \beta; n)$-robust combiner for every $\alpha, \beta \geq 0$ satisfying $\alpha + \beta \geq \delta$, where δ is some threshold, it might be the case that the combiner makes explicit use of the values α, β, and thus works differently for every particular values of α, β. In such a scenario more desirable would be a *uniform* construction, i.e. a single combiner that is secure under the sole assumption that $\alpha + \beta \geq \delta$. In particular, such a combiner wouldn't obtain the values of α, β as parameters. The desired properties of a uniform combiner are summarized in the following definition.

Definition 7 ($\{\delta$, n$\}$-robust uniform $\mathcal{A}$-combiner) *Let $\mathcal{A}$ be a two-party primitive. We say that an $\mathcal{A}$-combiner is a $\{\delta, n\}$-*robust uniform $\mathcal{A}$-combiner *if it is an $(\alpha, \beta; n)$-robust $\mathcal{A}$-combiner,* simultaneously *for all α and β satisfying $\alpha + \beta \geq \delta$.*

Notice that the parameter δ is a bound on *the sum* of the number of candidates secure for Alice and the number of candidates secure for Bob, hence given n candidates δ is from the range $0 \ldots 2n$. As an example consider a $\{4, 3\}$-robust *uniform* combiner. Such a combiner is a (regular) $(2; 3)$-robust combiner, but at the same time it is also a $(3, 1; 3)$-robust combiner and a $(1, 3; 3)$-robust combiner, i.e. it tolerates input candidates $\mathsf{C}_1, \mathsf{C}_2, \mathsf{C}_3$, where one candidate is secure for both parties, and the remaining two candidates are secure only for Alice resp. only for Bob. Furthermore, it is not difficult to see that not every $(k; n)$-robust combiner is automatically also a $\{\delta, n\}$-robust *uniform* combiner with $\delta = 2k$. In particular, the elegant $(2; 3)$-robust OT-combiner from [HKN^{+}05] breaks on inputs of the type described above for $(3, 1; 3)$-robust combiner.

3.4.2 OT-combiners with secure majority

The impossibility of transparent black-box (1; 2)-robust OT-combiners implies directly the impossibility of transparent black-box $(n; 2n)$-robust OT-combiners, as from the existence of the latter would follow the existence of the former. Similarly, it implies also the impossibility of transparent black-box $(\alpha, \beta; n)$-robust combiners for $\alpha + \beta \leq n$. However, since $(k, k; n)$-robust combiners are stronger than $(k; n)$-robust combiners, we can show very simple impossibility results, which essentially exclude $(\alpha, \beta; n)$-robust OT-combiners of any type[9] that would work for $\alpha+\beta \leq n$: in Lemma 4 we prove that there are no *black-box* OT-combiners with such robustness, and in Lemma 5 we show that constructing an OT-combiner of *any* type (for $\alpha + \beta \leq n$) is at least as hard as constructing an OT protocol without any assumptions.

As mentioned previously, these results are not directly comparable with the impossibility result from [HKN+05]: on one hand our results are stronger, since they go beyond *transparent black-box* combiners, but on the other hand they are weaker, since they exclude a primitive which is stronger then the one considered in [HKN+05].

Lemma 4 *There does not exist a black-box $(\alpha, \beta; n)$-robust OT-combiner for α, β, n satisfying $\alpha + \beta \leq n$.*

Proof. Assume that such a combiner exists, for some values α, β, and n, such that $\alpha + \beta \leq n$ holds. Let OT_1 be a trivial instance of OT where the sender sends both values to the receiver, and let OT_2 be a trivial instance where the receiver sends his choice bit to the sender, who sends back to the receiver the requested value. Observe that OT_1 is information-theoretically secure for the receiver, and OT_2 is information-theoretically secure for the sender.

Consider calling the combiner with input consisting of β instances of OT_1 and $n - \beta \geq \alpha$ instances of OT_2, and let $\overline{\mathsf{OT}}$ denote the resulting OT protocol. By assumption, $\overline{\mathsf{OT}}$ is secure for both parties. Since it is impossible to construct an OT protocol information-theoretically secure for *both* the sender and the receiver, there exists an adversary A (possibly inefficient), which breaks the protocol $\overline{\mathsf{OT}}$. By the definition of a black-box combiner, it follows that given oracle access to A one can break $2n - \alpha - \beta + 1 > n$ "sides" of the candidates. However, since one side of

[9] I.e., not only *transparent black-box* combiners.

each candidate is information-theoretically secure, we can break at most n sides. A contradiction. □

Lemma 5 *Any $(\alpha, \beta; n)$-robust OT-combiner for $\alpha + \beta \leq n$ implies the existence of OT.*

Proof. Assume that such a combiner exists, for some values α, β, and n such that $\alpha + \beta \leq n$. Let OT_1 and OT_2 be trivial instances of OT, as in the proof of Lemma 4. Calling the combiner using β instances of OT_1 and $n - \beta \geq \alpha$ instances of OT_2 as input yields a secure OT protocol without any assumption. □

We will now show that the bound of Lemmas 4 and 5 is tight, by presenting constructions of $(\alpha, \beta; n)$-robust OT-combiners for any α, β, and n satisfying $\alpha + \beta > n$. First we describe a combiner, which is very simple but not fully satisfactory, as it is not efficient.[10]

Lemma 6 *For every $\alpha \geq 0$, $\beta \geq 0$ and $\alpha + \beta > n$ there exists an* inefficient *third-party black-box $(\alpha, \beta; n)$-robust OT-combiner.*

Proof. The combiner is a straightforward generalization of the $(2; 3)$-robust OT-combiner from [HKN+05], which uses two special-purpose OT-combiners, combiner $\mathbf{R}$ for protecting the receiver, and combiner $\mathbf{S}$ for protecting the sender (cf. Section 3.1.4).

The $(\alpha, \beta; n)$-robust combiner works in two phases: in the first phase subsets of the candidates of size α are combined using the combiner $\mathbf{R}$, resulting in $n' = \binom{n}{\alpha}$ OT schemes. Each resulting instance is secure for the receiver and at least one is secure for both parties. In the second phase the n' OTs are combined using the combiner $\mathbf{S}$ to yield a final scheme protecting both the sender and the receiver. □

The combiner presented in the above proof is perfect in the sense that it does not introduce any additional error. However, it is inefficient in n, since the value of n', i.e., the number of OTs resulting from the first phase, would be superpolynomial in n. Lemma 8 presents a combiner that is not perfect, but efficient in n and other parameters, as required in Definition 6. In the construction we use a third-party combiner for bit commitment, which is an adaptation to our setting of a BC-combiner based on secret-sharing, due to Herzberg [Her05]. As this may be of independent interest, we describe it separately.

[10]Due to its inefficiency, this is strictly speaking not a robust combiner (cf. Def. 6). In a slight abuse of terminology, we call it an *inefficient* combiner.

Lemma 7 *For every $n \geq 2$ and for every $\alpha, \beta > 0$ satisfying $\alpha + \beta > n$ there exists a third-party black-box $(\alpha, \beta; n)$-robust BC-combiner.*

Proof. We describe a string commitment that allows a sender to commit to a value $s \in \{0,1\}^m$, for an arbitrary m satisfying $2^m > n$, using n candidate implementations of bit-commitment from which at least α are hiding and at least β are binding.

The sender computes[11] an ℓ-out-of-n Shamir's secret sharing of s over $\mathbb{F}_{2^m}$, for $\ell := n - \alpha + 1$, resulting in shares $s_1, \ldots, s_n$. Then the sender uses the n instances of bit-commitment to commit to the shares s_i bit-by-bit, for each $i \in [n]$. In the opening phase the sender opens the commitments to all the shares, and the receiver reconstructs the secret s.

To see that this commitment is hiding, notice that at least α shares are guaranteed to be hidden from the receiver, since at least α candidate bit-commitments are hiding. Hence before the opening phase the receiver sees at most $n - \alpha < \ell$ shares, which give no information about the secret. On the other hand, since at least β candidates of the bit-commitments are binding, the sender is indeed committed to at least β shares. Since we have $\alpha + \beta > n$, i.e., $\beta > n - \alpha$, the sharing polynomial, which has degree at most $n - \alpha$, is uniquely determined by these β shares, and so the commitment to s is also binding. $\square$

Lemma 8 *For every $n \geq 2$ and for every $\alpha, \beta > 0$ satisfying $\alpha + \beta > n$ there exists a third-party black-box $(\alpha, \beta; n)$-robust OT-combiner.*

Proof. First assume that the sender and the receiver have a common random string r at their disposal. Later we describe how this additional assumption can be dropped.

Using r, the combiner works as follows: it simulates (p,q)-WOT with $p + q \leq 1 - 1/n$ by picking each time an input OT-candidate uniformly at random. This is possible, since we are having n candidates, α of which are secure for the sender and β are secure for the receiver, with $\alpha + \beta \geq n + 1$. By picking one candidate at random we obtain a probability $p \leq (n - \beta)/n$ that the sender learns the receiver's choice, and a probability $q \leq (n - \alpha)/n$ that the receiver learns both bits input by the sender, hence $p + q \leq ((n-\alpha) + (n-\beta))/n = (2n - \alpha - \beta)/n \leq 1 - 1/n$, as required. Given such a (p,q)-WOT, use the (efficient) amplification algorithm of Damgård *et al.* [DKS99] to obtain a secure OT.

[11] In this computation we view bit-strings from $\{0,1\}^m$ as elements of $\mathbb{F}_{2^m}$.

To complete the argument, we have to show how the sender and the receiver can generate a common random string r. It is well-known that OT implies bit-commitment [Cré87], and bit-commitment implies coin-toss [Blu82]. Therefore, we can convert our n candidate implementation of OT into n candidate implementations of bit-commitment, and then use the bit-commitment-combiner of Lemma 7 to obtain a secure implementation of bit-commitment, provided that $\alpha + \beta > n$. This implementation can then be used to implement a coin-toss, i.e., the parties can generate a common random string r using the input candidates only, without additional assumptions. Finally, it is easy to verify that all the described protocols use the candidates in a third-party black-box manner, and that the combiner is efficient. □

From Lemmas 4 and 8, we get immediately the following theorem about $(\alpha, \beta; n)$-robust combiners of oblivious transfer.

Theorem 7 *There exists a black-box $(\alpha, \beta; n)$-robust OT-combiner if and only if $\alpha + \beta > n$ holds. The construction is third-party black-box and efficient.*

Furthermore, the impossibility result from [HKN$^+$05] together with Lemma 8 yield the following corollary about $(k; n)$-robust OT-combiners.

Corollary 6 *There exists a transparent black-box $(k; n)$-robust OT-combiner if and only if $2k > n$. The construction is third-party black-box and efficient.*

3.4.3 OT-combiners based on the symmetry of OT

A closer look at the combiners from the proofs of Lemmas 6 and 8 shows that these are "non-uniform" combiners (cf. Sect. 3.4.1). Namely, the proofs show that for given $\alpha, \beta > 0$ with $\alpha + \beta > n$ there exists a $(\alpha, \beta; n)$-robust combiner, i.e., the actions of the combiner are different for different values α, β. (For Lemma 6 it is explicit in the construction, and for Lemma 8 it is due to the fact that the amplification algorithm from [DKS99] makes explicit use of parameters p, q.) More desirable would be an *uniform* construction, which would have a switched order of quantifiers, i.e., we would like a single combiner that is secure for every $\alpha, \beta > 0$ with $\alpha + \beta > n$, and would therefore be *strictly stronger* than any of the special combiners. In this section we show how to construct such a combiner by exploiting the *symmetry* of OT, i.e., the fact that given OT with sender Alice and receiver Bob, we can perfectly *logically* reverse it to obtain OT with receiver Alice and sender Bob [CS91, OVY93, WW06].

Our construction is based on a simple trick, which is somehow non-standard, yet plausible in many scenarios: we require that the parties can *swap their roles* when executing candidate protocols, i.e., any input candidate OT_i can be executed in such a way that the sender (of the main OT-protocol) plays the role of receiver in OT_i, and the receiver plays the role of sender in OT_i. Such a swapping of the roles in an execution of a protocol can be viewed as a *physical* reversal of the protocol.

Moreover, we require that we have at our disposal *multiple copies of each candidate implementation* (in particular, our protocols use the candidates both in the original setting as well in the swapped configuration). For example, if the input candidates are given as software packages, these requirements are not a problem, as it means only calling different functions, but if a candidate is given as a pair of physical devices implementing the primitive, the swapping operation can be problematic, as it may require real physical swap of the corresponding devices. However, it is difficult to come up with primitive that cannot be swapped or duplicated *in principle*. Such a primitive would need to make use of some kind of a physical phenomenon, only available to one of the parties, but not to the other.

We use the swapping of the roles in OT, i.e., a physical reversal of OT, together with a logical reversal of OT [WW06] to obtain an OT in the original direction (from the original sender to the original receiver), but *with swapped security properties*. More precisely, let **swap** be this two-step process, that is a physical swap followed by a logical reversal, and consider an implementation OT and its swapped-and-reversed version, $\mathsf{OT}^* = \texttt{swap}(\mathsf{OT})$. If OT is a correct OT-protocol, then so is OT^*. Moreover, if in OT the security of the sender is based on some assumption $\mathcal{A}$, and the security of the receiver is based on some assumption $\mathcal{B}$, then in OT^* we have the opposite situation: the security of the sender relies on assumption $\mathcal{B}$, and the security of the receiver relies on assumption $\mathcal{A}$. In particular, if OT is an implementation unconditionally secure for the sender, then OT^* is an implementation unconditionally secure for the receiver.

As a first application of this swapping trick we show a $\{3,2\}$-robust *uniform* OT-combiner, i.e., a combiner which is simultaneously $(\alpha, \beta; 2)$-robust for any α, β satisfying $\alpha + \beta \geq 3$. Recall that if it is known in advance that the security of one party is guaranteed (e.g. $\alpha = 2$), then the corresponding combiner is very simple [HKN+05]. However, the combiner for the case $\alpha = 2$ is quite different from the combiner for the case $\beta = 2$, hence these simple combiners are not uniform.

SENDER'S INPUT: two bits b_0, b_1

RECEIVER'S INPUT: choice bit c

INPUT OT PROTOCOLS: $\mathsf{OT}_1, \mathsf{OT}_2$

Note: The combiners $\mathbf{R}$ and $\mathbf{S}$ are described Sect. 3.1.4.

1. Parties apply **swap** to obtain

$$\mathsf{OT}_1^* := \mathbf{swap}(\mathsf{OT}_1) \text{ and } \mathsf{OT}_2^* := \mathbf{swap}(\mathsf{OT}_2) \ .$$

2. Parties define $\mathsf{OT}' := \mathbf{R}(\mathsf{OT}_1, \mathsf{OT}_2)$ and $\mathsf{OT}'' := \mathbf{R}(\mathsf{OT}_1^*, \mathsf{OT}_2^*)$.
3. Parties invoke $\mathbf{S}(\mathsf{OT}', \mathsf{OT}'')(b_0, b_1; c)$.

Figure 3.5: A $\{3, 2\}$-robust *uniform* OT-combiner.

The idea behind our uniform combiner is to use *both*, the two candidate OT_1, OT_2, and their swapped counterparts $\mathsf{OT}_1^* = \mathbf{swap}(\mathsf{OT}_1)$ and $\mathsf{OT}_2^* = \mathbf{swap}(\mathsf{OT}_2)$. Since $\alpha + \beta \geq 3$, at least two of OT_1, OT_2, OT_1^*, OT_2^* are secure for both parties, at most one is insecure for the sender, and at most one is insecure for the receiver. This is sufficient to implement a secure OT. The construction makes use of the two special-purpose OT-combiners from [HKN+05], namely combiner $\mathbf{S}$ for protecting the sender, and combiner $\mathbf{R}$, which protects the receiver (cf. Sect. 3.1.4). Figure 3.5 presents the entire construction in more detail, and the following theorem summarizes its properties.

Theorem 8 *There exists a third-party black-box {3, 2}-robust uniform OT-combiner using the* **swap***-operation.*

Proof. *(sketch)* Consider the protocol in Figure 3.5. Let $\overline{\mathsf{OT}}$ denote the resulting OT protocol. $\overline{\mathsf{OT}}$ has to satisfy correctness, privacy for the sender, and privacy for the receiver. Correctness is trivially given due to correctness of the candidates OT_1, OT_2, the symmetric schemes OT_1^*, OT_2^*, and the combiners $\mathbf{R}$ and $\mathbf{S}$. Given the symmetry of OT, if the privacy of one party is compromised for one candidate, then the privacy of the other party is compromised for the corresponding swapped candidate. Combining OT_1, OT_2, respectively OT_1^*, OT_2^*, with $\mathbf{R}$ ensures that receiver's privacy is protected in both OT' and OT'', and sender's privacy in at least one of them. Hence $\mathbf{S}(\mathsf{OT}', \mathsf{OT}'')$ protects the sender from a possible security break of one of the input candidates. Finally, is easy to verify that this is a third-party black-box combiner. □

The next lemma gives a general construction for obtaining a uniform combiner from a non-uniform one. This construction makes use of the **swap**-operation, and can be used for combiners of any symmetric two-party primitive.

Lemma 9 *If there exists a $(k, k; 2n)$-robust OT-combiner, then there exists a $\{k, n\}$-robust uniform OT-combiner using the* **swap***-operation.*

Proof. *(sketch)* The $\{k, n\}$-robust uniform OT-combiner works as follows: given n candidate instances of OT, satisfying $\alpha + \beta \geq k$, we duplicate all instances, and apply the **swap**-operation to the duplicates. In this way we obtain $2n$ candidate instances, and at least k of them are secure for the sender, and at least k are secure for the receiver. Now we can apply the $(k, k; 2n)$-robust OT-combiner to these $2n$ instances, and get a secure implementation of OT. □

Lemma 9 together with Theorem 7, give us the following theorem.

Theorem 9 *For any $n \geq 2$ and $\delta > n$, there exists a third-party black-box $\{\delta, n\}$-robust uniform OT-combiner using the* **swap***-operation.*

3.5 Robust combiners for OLFE

In this section, we propose two constructions of robust combiners for *oblivious linear function evaluation* (OLFE), a primitive which is a generalization of oblivious transfer (cf. Sect. 3.1.1). In particular, we present a $(\alpha, \beta; n)$-robust OLFE-combiner works for any $\alpha + \beta > n$. The proposed construction achieves optimal robustness and is much more efficient than the most efficient OT-combiners with the same robustness [MPW07]: it uses any candidate instance only *once*, which is optimal, and therefore might be preferable in some scenarios. For example, if we have a protocol for secure function evaluation based on OLFE, a protocol for OLFE based on OT, and three implementations of OT from which we assume two to be correct, it will be more efficient to use the OLFE-combiner we present in this section than any known OT-combiner: the construction would only require only half as many calls to each OT-instance. Furthermore, the construction is perfect (its error probability is equal zero). In the second construction presented in this section we exploit the symmetry of OLFE to obtain an efficient *uniform* OLFE-combiner.

3.5.1 OLFE-combiner

Recall that OLFE is a primitive defined over some finite field $\mathbb{F}_q$, where the sender's input is a linear function $f(x) = a_1x + a_0$, with $a_0, a_1, x \in \mathbb{F}_q$, and the receiver's input is an argument $c \in \mathbb{F}_q$. The receiver learns only the value of the function on his input, $y = f(c)$, and the sender learns nothing.

In our OLFE-combiner we use Shamir secret sharing scheme [Sha79] for both players at the same time, to protect the privacy of their inputs. Given inputs $f(x) := a_1x + a_0$ resp. $c \in \mathbb{F}_q$, the sender and the receiver proceed as follows. The sender picks two random polynomials $A_0(z)$ of degree $n - 1$ and $A_1(z)$ of degree $n - \alpha$, such that $A_0(0) = a_0$ and $A_1(0) = a_1$. The receiver picks a random polynomial $C(z)$ of degree $n-\beta$, such that $C(0) = c$. Then the parties evaluate locally these polynomials for n distinct non-zero values $z_1, \ldots, z_n \in \mathbb{F}_q$, and use the resulting values as input to the instances of OLFE.

More precisely, we can view the two polynomials of the sender, $A_0(z)$ and $A_1(z)$ as parts of a two-dimensional polynomial $F(x, z)$ of degree 1 in x and degree $n - 1$ in z:

$$F(x, z) := A_1(z) \cdot x + A_0(z) \ ,$$

which satisfies

$$F(x, 0) = f(x) \ .$$

Note that for any constant $z_i \in \mathbb{F}_q$, the polynomial $f_i(x) := F(x, z_i) = A_1(z_i)x + A_0(z_i)$ is just a linear function. Furthermore we define a polynomial $h(z)$

$$h(z) := F(C(z), z) = A_1(z) \cdot C(z) + A_0(z) \ ,$$

which clearly satisfies

$$h(0) = f(c) \ ,$$

i.e., $h(0)$ is the value the receiver should obtain. Hence the goal is now to allow the receiver to learn $h(0)$. To achieve this, the parties run OLFE

Protocol **`OLFE-rc`**

SENDER'S INPUT: linear function $f(x) := a_1 x + a_0$, with $a_0, a_1 \in \mathbb{F}_q$

RECEIVER'S INPUT: evaluation point $c \in \mathbb{F}_q$

INPUT OLFE PROTOCOLS: $\mathsf{OLFE}_1, \ldots, \mathsf{OLFE}_n$

parameters: $n < q; \alpha, \beta$; distinct non-zero constants $z_1, \ldots, z_n \in \mathbb{F}_q$

1. Sender picks two random polynomials:

   $A_0(z)$ of degree $n-1$ such that $A_0(0) = a_0$, and
   $A_1(z)$ of degree $n-\alpha$ such that $A_1(0) = a_1$.

2. Receiver picks a random polynomial $C(z)$ of deg. n–β, s.t. $C(0) = c$.
3. $\forall i \in [n]$ the parties run OLFE_i, with sender holding input $f_i(x) = A_1(z_i)x + A_0(z_i)$, and receiver holding input $c_i = C(z_i)$.
4. Receiver uses the values $\{(z_1, f_1(c_1), \ldots, (z_n, f_n(c_n))\}$ to interpolate a polynomial $\tilde{h}(z)$ of degree $n-1$, and outputs $y = \tilde{h}(0)$.

Figure 3.6: `OLFE-rc`: $(\alpha, \beta; n)$-robust OLFE-combiner for $\alpha+\beta>n$.

candidates, through which the receiver obtains sufficiently many points on $h(z)$ to enable its interpolation and the computation of $h(0)$. To evaluate the polynomial $h(z)$ through OLFE_i at any particular value $z_i \in \mathbb{F}_q$, the sender's input is $f_i(x) := F(x, z_i)$, and the receiver's input is $C(z_i)$.

Intuitively, the privacy of the users in this construction is protected by the degrees of the polynomials we use. Since we have to be prepared that in worst case only α of the n input OLFE-protocols are secure for the sender, and only β are secure for the receiver, the polynomials $A_0(z)$ and $A_1(z)$ must have degree at least $n-\alpha$, and the degree of $C(z)$ must be at least $n-\beta$.

On the other hand, note that $h(z)$ is of degree $\max\{\deg(A_0), \deg(A_1) + \deg(C)\} = \max\{n-1, 2n-\alpha-\beta\}$. Since we're using n evaluation points, this degree must be at most $n-1$, otherwise interpolation is not possible. This implies, that $\alpha+\beta > n$ must be satisfied. Indeed, this construction works for any α, β with $\alpha+\beta > n$, which is optimal. Figure 3.6 presents the combiner in full detail, and its formal analysis is given below.

Lemma 10 *Protocol* **`OLFE-rc`** *is correct if $\alpha + \beta > n$.*

Proof. By construction, we have $f(x) = F(x, 0)$ and $c = C(0)$. Hence $f(c) = F(C(0), 0) = h(0)$. Further, we have $y_i := f_i(c_i) = f(g(z_i), z_i) = h(z_i)$. Since h has degree $\max\{n-1, 2n-\alpha-\beta\} < n$, the interpolation from y_i's will result in a polynomial $\tilde{h}(z)$ identical to $h(z)$. Therefore, we have $y = \tilde{h}(0) = h(0) = f(c)$. □

Lemma 11 *Protocol* **`OLFE-rc`** *is secure against a malicious sender if at least β input instances of OLFE are secure against a malicious sender.*

Proof. Note that the values c_i form a $(n-\beta+1)$-out-of-n secret sharing of c, hence a malicious sender that sees at most $(n-\beta)$ values of c_i (of his choice) does not get any information about c. □

Lemma 12 *Protocol* **`OLFE-rc`** *is secure against a malicious receiver if at least α input instances of OLFE are secure against a malicious receiver.*

Proof. It is sufficient to prove that the receiver does not get any information about a_1. The receiver will use arbitrary values c_i and obtain $f_i(c_i)$. Additionally, he will receive sender's entire input in at most $n-\alpha$ runs of OLFE, namely for those instances of OLFE_j whose security is broken. Since he has already received one value on each $f_i(z)$, this is equivalent to providing him additionally with $A_1(z_j)$ for at most $n-\alpha$ values of $j \in [n]$. We will show that for any value $\overline{a}_1$, there exists exactly one possible sharing of the values consistent with the receivers view, which proves that he does not get information. Given $\overline{a}_1$ and the values $A_1(z_j)$, we can interpolate the unique polynomial $\overline{A}_1(z)$ of degree $n-\alpha$, and for the values $f_i(c_i) - \overline{A}_1(z)c_i$, we can interpolate the unique polynomial $\overline{A}_0(z)$ of degree $n-1$. □

From the above lemmas, and from lower bounds on robustness of OT-combiners presented in the previous section we obtain the following theorem:

Theorem 10 *For every $n > 1$ and α, β satisfying $\alpha + \beta > n$ there exists an efficient third-party black-box $(\alpha, \beta; n)$-robust combiner for OLFE over $\mathbb{F}_q$ with $q > n$. The combiner is perfect, achieves optimal robustness, and uses only one run of each candidate instance of OLFE, which is also optimal.*

3.5.2 Uniform OLFE-combiner based on symmetry

Recall that by using the symmetry of OT we were able to construct efficient construct efficient *universal* OT-combiners (cf. Sect. 3.4.3). Such combiners are based on an observation that a physical reversal of a symmetric primitive followed by a logical reversal yields an implementation with swapped security properties. Recall also, that by **`swap`** we denote this two-step process, i.e. a physical swap followed by a logical reversal. Since OLFE is also symmetric [WW06], we obtain immediately the following theorem.

Theorem 11 *For any $n > 1$ and any $\delta > n$ there exists a third-party black-box $\{\delta, n\}$-robust uniform combiner for OLFE over $\mathbb{F}_q$ with $q > 2n$, using the* **`swap`**-*operation. The combiner is perfect and uses only two runs of each candidate instance of OLFE.*

Chapter 4

Conclusions

4.1 Asynchronous multi-party computation

In the first part of this thesis we have proposed a secure multi-party computation protocol which advances both theory and practice in this field. From a theoretical point of view, the protocol is optimally resilient, fully asynchronous, and has asymptotically lower communication complexity than any previous asynchronous MPC protocol. The protocol communicates only $\mathcal{O}(n^2\kappa)$ bits per multiplication gate, which is only by a factor $\mathcal{O}(n)$ worse than the most efficient known protocol for *synchronous* networks [HN06]. Moreover, the proposed protocol requires very few invocations of the broadcast primitive (independent of the size of the computed circuit).

From a practical point of view, the new protocol is designed for real-world networks with unknown message delay, allows every party to provide his input under a very reasonable assumption (few rounds of synchronization), and achieves best-possible resilience against cheating (up to a third of the parties may misbehave). Furthermore, the protocol is very efficient, the constant communication overhead is minimal, and the constants hidden in the $\mathcal{O}(\cdot)$-notation are small. The effective evaluation of the circuit takes less than $20\,n^2\kappa$ bits of communication per multiplication, which makes the protocol applicable for reasonably sized circuits among small sets of parties. The key set-up (for the threshold decryption and threshold signatures) is more communication-intensive; however, this can be performed long in advance.

At the first sight the existence of the gap between the bit complexities of the proposed MPC protocol and the recent protocol of [HN06] may seem unsatisfactory, but we stress that our protocol works in the asynchronous model, which is a better model for real-world networks. Moreover, the synchronous protocol was proposed only very recently, and in fact uses some of the techniques proposed in this thesis. This gives some hope that further reduction of communication complexity might be possible also in the asynchronous case. Indeed, we believe, that one can obtain linear asynchronous communication, but it seems that such a protocol will require some new idea(s). One difficulty in a adapting the ideas of [HN06] to the asynchronous model stems from the fact, that the protocol of [HN06] makes use of the *player elimination* technique [HMP00], which relies on synchronization.

Another drawback of our protocol is its limitation to static adversaries. Although there is no (obvious) adaptive attack and the limitation is more due to the lack of formal simulatability in presence of an adaptive adversary, it would be nice (at least for theoretical reasons) to have a provably *adaptively* secure protocol.

4.2 Robust combiners

In the second part of this thesis we studied the possibility of increasing robustness of cryptographic protocols by using robust combiners. From the practical side, such constructions can be viewed as a generic way of basing security of cryptographic systems on multiple assumptions, and offer protection against wrong assumptions about the difficulty of computational problems. From the theory point of view, combiners can be considered as generalized reductions, and serve as a tool for studying relations between primitives. In particular, we have presented constructions of $(1; 2)$-robust PIR-combiners, and also cross-primitive combiners: PIR-to-BC and PIR-to-OT. The existence of simple and efficient PIR-combiners is somewhat surprising given the impossibility result for OT-combiners. Moreover, the proposed PIR-to-OT combiner rules out certain types of reductions of PIR to OT.

A closer look at the PIR-to-BC and PIR-to-OT combiners reveals a common theme — we use a reduction of the target primitive to the input primitive, and exploit additional security properties guaranteed by the reduction to obtain an efficient combiner. Motivated by this observation, we proposed stronger definitions for robust combiners for two-party protocols, which yield robuster, more general combiners for oblivious transfer and other primitives. Specifically, we presented *uniform* combiners for OT and OLFE, which tolerate a wide spectrum of insecure input candidates.

As we mentioned earlier, despite of many implicit applications of combiners in the literature, a more formal study of such constructions has been initiated quite recently, which indicates that many questions in this area haven't been explored yet. We close this section with a few examples of open problems closely related to the presented work:

- While the basic PIR-combiner we propose is applicable to many PIR protocols described in the literature, it is not universal in the sense that it does not work for *any non-trivial* PIR schemes — the combiners requires one *two-message* PIR and some bounds on communication complexities. It would be interesting to either find an universal combiner that does not need such assumptions or to further optimize the current combiner while maintaining its applicability.

- An intermediate step towards universal $(1;2)$-robust PIR-combiners might be a construction of an universal $(2;3)$-robust PIR-combiner. Oblivious transfer and bit commitment, primitives considered to be hard to combine with $(1;2)$-robust combiners, do have very efficient universal $(2;3)$-robust combiners.

- With regard to cross-primitive combiners, we have argued that there exists a PIR-to-BC combiner which yields statistically hiding bit commitment, and that there exists one yielding statistically binding bit commitment. However, for the later only an inefficient, generic construction via a combiner for one-way functions is known. It would be interesting to find a more efficient, direct PIR-to-BC combiner yielding a statistically binding bit commitment scheme.

- Although the proposed uniform OT-combiner works with all OT protocols described in the literature, it would be interesting to know whether the used role-swapping technique can be dropped. In addition, there is currently a trade-off between *perfect* security and efficiency: we don't know whether there is an efficient (in the number of candidates) perfect uniform OT-combiner.

Bibliography

[**Note:** *the numbers after each item denote pages, on which the item was referenced.*]

[AB81] C.A. Asmuth and G.R. Blakely. An effcient algorithm for constructing a cryptosystem which is harder to break than two other cryptosystems. *Computers and Mathematics with Applications*, 7:447–450, 1981. 6

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In *Proc. 8th ACM PODC*, pages 201–210, 1989. 4

[BB06] Dan Boneh and Xavier Boyen. On the impossibility of efficiently combining collision resistant hash functions. In *Proc. CRYPTO '06*, pages 570–583, August 2006. 6

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proc. 25th ACM STOC*, pages 52–61, 1993. 4, 43

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Proc. CRYPTO '91*, pages 420–432, 1991. 44, 45

[Bec54] Samuel Beckett. *Waiting for Godot*. New York: Grove Press, 1954. 4

[BFKR90] Donald Beaver, Joan Feigenbaum, Joe Kilian, and Phillip Rogaway. Security with low communication overhead (extended abstract). In *Proc. CRYPTO '90*, pages 62–76, 1990. 4

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM STOC*, pages 1–10, 1988. 1, 2, 4

[BIKM99] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. One-way functions are essential for single-server private information retrieval. In *Proc. ACM STOC*, pages 89–98, 1999. 7, 58, 70, 71, 73, 74

[BKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience. In *Proc. 13th ACM PODC*, pages 183–192, 1994. 4, 43

[BKY03] Daniel Bleichenbacher, Aggelos Kiayias, and Moti Yung. Decoding of interleaved Reed-Solomon codes over noisy data. In *Proc. ICALP 2003*, pages 97–108, 2003. 61

[Bla79] George R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, pages 313–317. American Federation of Information Processing Societies, 1979. 62

[Blu81] Manuel Blum. Coin flipping by telephone. In *Proc. CRYPTO '81*, pages 11–15, 1981. 57

[Blu82] Manuel Blum. Coin flipping by telephone - a protocol for solving impossible problems. In *COMPCON, Proc. Twenty-Fourth IEEE Computer Society International Conference*, pages 133–137, 1982. 57, 81

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proc. 22nd ACM STOC*, pages 503–513, 1990. 4, 51

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. ACM Conference on Computer and Communications Security*, pages 62–73, 1993. 14, 18, 22

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures — How to sign with RSA and Rabin. In *Proc. EUROCRYPT '96*, pages 399–416, 1996. 18

[BT85] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, 1985. 8

[Can95] Ran Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995. 4

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000. 12

[CC06] Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In *Proc. CRYPTO '06*, pages 521–536, 2006. 5

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. 20th ACM STOC*, pages 11–19, 1988. 1, 2, 4

[CCG$^+$07] Hao Chen, Ronald Cramer, Shafi Goldwasser, Robbert de Haan, and Vinod Vaikuntanathan. Secure computation from random error correcting codes. In *Proc. EUROCRYPT '07*, 2007. to appear. 5

[CDD$^+$99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Proc. EUROCRYPT '99*, pages 311–326, 1999. 4

[CDD00] Ronald Cramer, Ivan Damgård, and Stefan Dziembowski. On the complexity of verifiable secret sharing and multiparty computation. In *Proc. 32nd ACM STOC*, pages 325–334, 2000. 5

[CDF01] Ronald Cramer, Ivan Damgård, and Serge Fehr. On the cost of reconstructing a secret, or vss with optimal reconstruction phase. In *Proc. CRYPTO '01*, pages 503–523, 2001. 4

[CDG$^+$05] Ronald Cramer, Vanesa Daza, Ignacio Gracia, Jorge Jiménez Urroz, Gregor Leander, Jaume Martí-Farré, and Carles Padró. On codes, matroids and secure multi-party computation from linear secret sharing schemes. In *Proc. CRYPTO '05*, pages 327–343, 2005. 5

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Proc. TCC '05*, pages 342–362, 2005. 4

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Proc. EUROCRYPT '00*, pages 316–334, 2000. 5

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Proc. EUROCRYPT '01*, pages 280–300, 2001. 4, 8, 11, 16, 18, 19, 36, 39, 51

[CG88] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988. 71

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proc. 30th ACM STOC*, pages 209–218, 1998. 14, 18

[Cha04] Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In *Proc. Information Security and Privacy: 9th Australasian Conference, ACISP 2004*, pages 50–61, 2004. 7

[CK88] Claude Crépeau and Joe Kilian. Achieving oblivious transfer using weakened security assumptions (extended abstract). In *Proc. IEEE FOCS '88*, pages 42–52, 1988. 57, 62

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998. 7, 56, 69

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinopole: Practical asynchronous Byzantine agreement using cryptography. In *Proc. 19th ACM PODC*, pages 123–132, 2000. 18, 41

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Proc. EUROCRYPT '99*, pages 402–414, 1999. 6, 7, 61

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proc. 25th ACM STOC*, pages 42–51, 1993. 14

[Cré87] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In *Proc. CRYPTO '87*, pages 350–354, 1987. 57, 81

[CS91] Claude Crépeau and Miklós Sántha. On the reversibility of oblivious transfer. In *Proc. EUROCRYPT '91*, pages 106–113. Springer, 1991. 81

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. 1

[DIO01] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Universal service-providers for private information retrieval. *Journal of Cryptology*, 14(1):37–74, 2001. 7, 64

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Proc. 4th PKC*, pages 110–136, 2001. 18

[DK05] Yevgeniy Dodis and Jonathan Katz. Chosen-ciphertext security of multiple encryption. In *Proc. TCC '05*, pages 188–209, 2005. 6

[DKS99] Ivan Damgård, Joe Kilian, and Louis Salvail. On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In *Proc. EUROCRYPT '99*, pages 56–73, 1999. 57, 80, 81

[DMO00] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *Proc. EUROCRYPT '00*, pages 122–138, 2000. 7, 54, 73, 74, 75

[EG85] Shimon Even and Oded Goldreich. On the power of cascade ciphers. *ACM Trans. Comput. Syst.*, 3(2):108–116, 1985. 6

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985. 57, 73

[FH96] Matt Franklin and Stuart Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232, 1996. 8, 11, 18

[Fis02] Marc Fischlin. On the impossibility of constructing non-interactive statistically-secret protocols from any trapdoor one-way function. In *Proc. CT-RSA*, pages 79–95, 2002. 7

[FPS00] Pierre-Alain Fouque, Gouillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Proc. Financial Cryptography '00*, 2000. 18

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proc. CRYPTO '86*, pages 186–194, 1986. 22

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation. In *Proc. 24th ACM STOC*, pages 699–710, 1992. 4

[Gas04] William I. Gasarch. A survey on private information retrieval (column: Computational complexity). *Bulletin of the EATCS*, 82:72–107, 2004. 7

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. 1

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM STOC*, pages 218–229, 1987. 1, 2, 4

[Gol04] Oded Goldreich. *The Foundations of Cryptography*, volume II, Basic Applications. Cambridge University Press, 2004. 73

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proc. 17th ACM PODC*, 1998. 4

[Hai04] Iftach Haitner. Implementing oblivious transfer using collection of dense trapdoor permutations. In *Proc. TCC'04*, pages 394–409, 2004. 73

[Her05] Amir Herzberg. On tolerant cryptographic constructions. In *CT-RSA*, pages 172–190, 2005. full version on Cryptology ePrint Archive, eprint.iacr.org/2002/135. 6, 79

[HKN$^+$05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In *Proc. EUROCRYPT '05*, pages 96–113, 2005. 6, 9, 10, 54, 55, 58, 60, 61, 62, 70, 73, 75, 77, 78, 79, 81, 82, 83

[HL05] Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In *Proc. TCC '05*, pages 264–282, 2005. 6

[HM01] Martin Hirt and Ueli Maurer. Robustness for free in unconditional multi-party computation. In *Proc. CRYPTO '01*, pages 101–118, August 2001. 4

[HMP00] Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In *Proc. ASIACRYPT '00*, pages 143–161, December 2000. 4, 90

[HN06] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In *Proc. CRYPTO '06*, Lecture Notes in Computer Science, August 2006. 4, 8, 43, 89, 90

[HNP05] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In *Proc. EUROCRYPT '05*, pages 322–340, 2005. 9, 12

[HNP06] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication, 2006. manuscript in preparation. 9, 12

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proc. ACM STOC*, pages 44–61, 1989. 73

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th ACM STOC*, pages 20–31, 1988. 6

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proc. IEEE FOCS '00*, pages 364–373, 1997. 7, 56, 68

[KO00] Eyal Kushilevitz and Rafail Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In *Proc. EUROCRYPT '00*, pages 104–121, 2000. 7

[KY01] Aggelos Kiayias and Moti Yung. Secure games with polynomial expressions. In *Proc. ICALP 2001*, pages 939–950, 2001. 6, 61

[Lip05] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *Proc. Information Security, 8th International Conference, ISC 2005*, pages 314–328, 2005. 7

[Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, 1978. 1

[MM93] Ueli Maurer and James L. Massey. Cascade ciphers: The importance of being first. *Journal of Cryptology*, 6(1):55–61, 1993. preliminary version in *Proc. IEEE Symposium on Information Theory*, 1990. 6

[MP06] Remo Meier and Bartosz Przydatek. On robust combiners for private information retrieval and other primitives. In *Proc. CRYPTO '06*, pages 555–569, August 2006. 10, 55

[MPW07] Remo Meier, Bartosz Przydatek, and Jürg Wullschleger. Robuster combiners for oblivious transfer. In *Proc. TCC '07*, pages 404–418, 2007. 10, 55, 84

[MRH04] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Proc. TCC '04*, pages 21–39, 2004. 14, 18

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991. 70, 75

[Nie02] Jesper B. Nielsen. A threshold pseudorandom function construction and its applications. In *Proc. CRYPTO '02*, pages 401–416, 2002. 18

[OVY93] Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Fair games against an all-powerful adversary. In *Advances in Computational Complexity Theory*, volume 13 of *AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 155–169. AMS, 1993. 81

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. EUROCRYPT '99*, pages 223–238, 1999. 18

[Pie07] Krzysztof Pietrzak. Non-trivial black-box combiners for collision-resistant hash-functions don't exist. In *Proc. EUROCRYPT '07*, 2007. To appear, available at eprint.iacr.org/2006/348. 6

[PSR02] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In *Proc. Indocrypt 2002*, pages 93–107, 2002. 4, 5, 8, 43

[PW06] Bartosz Przydatek and Jürg Wullschleger. Error-tolerant combiners for oblivious primitives, 2006. Submitted manuscript. 10, 55

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer., 1981. Tech. Memo TR-81, Aiken Computation Laboratory, available at eprint.iacr.org/2005/187. 56, 57

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. 1, 18

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979. 61, 62, 85

[Sho00] Victor Shoup. Practical threshold signatures. In *Proc. EUROCRYPT '00*, pages 207–220, 2000. 18

[SR00] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In *Proc. Indocrypt 2000*, pages 117–129, 2000. 4, 5, 8, 43

[ST04] Berry Schoenmakers and Pim Tuyls. Practical two-party computation based on the conditional gate. In *Proc. ASIACRYPT '04*, pages 119–136, 2004. 50

[Tou84] Sam Toueg. Randomized byzantine agreements. In *Proc. 3rdACM PODC*, pages 163–178, 1984. 8

[WW06] Stefan Wolf and Jürg Wullschleger. Oblivious transfer is symmetric. In *Proc. EUROCRYPT '06*, pages 222–232, 2006. 81, 82, 88

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Proc. CRYPTO '05*, pages 17–36, 2005. 2

[WYY05b] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In *Proc. CRYPTO '05*, pages 1–16, 2005. 2

[Yao82] Andrew C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE FOCS*, pages 160–164, 1982. 1, 2, 4

Index