IST-2002-507932

ECRYPT

European Network of Excellence in Cryptology

Network of Excellence

Information Society Technologies

D.PROVI.9

Summary Report on Secure Computation Protocols

Due date of deliverable: 31. January 2008
Actual submission date: 12. February 2008

Start date of project: 1. February 2004
Duration: 4 years

Lead contractor: TUE

Revision 1.0

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>PU</th>
<th>PP</th>
<th>RE</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restricted to other programme participants (including the Commission services)</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restricted to a group specified by the consortium (including the Commission services)</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Confidential, only for members of the consortium (including the Commission services)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Summary Report on Secure Computation Protocols

Editor
Berry Schoenmakers (TUE)

Contributors
Christian Cachin (IBM),
Jan Camenisch (IBM),
Ivan Damgård (BRICS),
Paolo D’Arco (UNISA),
Olivier de Marneffe (UCL),
Klaus Kursawe (KUL),
Mark Manulis (UCL),
Jesper Buus Nielsen (BRICS),
Gregory Neven (KUL),
Svetla Nikova (KUL),
Giuseppe Persiano (UNISA),
Ahmad-Reza Sadeghi (RUB),
Berry Schoenmakers (TUE),
José Villegas (TUE),
Ivan Visconti (UNISA)

12. February 2008
Revision 1.0

The work described in this report has in part been supported by the Commission of the European Communities through the IST program under contract IST-2002-507932. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.
Abstract

This report summarizes the state of the art and open problems in the area of secure multiparty computation protocols. The report is divided into three main chapters. The first main chapter briefly discusses some common (mostly cryptographic) primitives used in multiparty computation protocols, namely broadcast channels, secret sharing, threshold cryptography, and oblivious transfer. Our current knowledge in these areas is summarized. The remaining two chapters then continue with the presentation of some specifically active research areas in secure multiparty computation, and state important open problems in each of these areas. Here, the second main chapter treats generic protocols for secure multiparty computation, while the third main chapter considers several practical scenarios and specific protocols for them.
Contents

1 Introduction 1

2 Primitives 3
  2.1 Fault-tolerant Broadcast 3
  2.2 Secret Sharing Schemes 4
  2.3 Threshold Cryptography 6
    2.3.1 Models 6
    2.3.2 Threshold Signature Schemes 8
    2.3.3 Threshold Pseudorandomness 8
    2.3.4 Threshold Key Generation 9
  2.4 Oblivious Transfer 9

3 Generic protocols 13
  3.1 General Secure Two-Party Computation 13
  3.2 The Generic MPC Problem 14
  3.3 Formalizing the Problem 14
    3.3.1 The early feasibility results 15
  3.4 The Typical Setting 16
  3.5 State of the Art and Open Problems 17
    3.5.1 The VSS Strain 17
    3.5.2 The Threshold-Encryption Strain 17
    3.5.3 Adaptive Security 18
    3.5.4 Asynchronous Security 19
  3.6 Aside: Strongly Multiplicative Linear Secret Sharing 19

4 Specific Applications and Multiparty Computation Protocols 21
  4.1 Fair exchange 21
  4.2 Private Matching 22
    4.2.1 Keyword search 24
  4.3 Anonymous Credentials 24
  4.4 Property-Based Attestation 26
  4.5 Group Key Exchange 29
  4.6 Multiparty Computation and Economy 30
    4.6.1 Auctions 31
    4.6.2 Fair Division 31
  4.7 Electronic Voting 32
Chapter 1

Introduction

In modern cryptography there is a whole body of work that deals with problems generally referred to as secure multiparty computation. The common setting is that two or more parties wish to execute a task jointly, where each of the parties will contribute private input values (possibly at several stages in the computation). All parties should be convinced that the task is performed properly, and all parties should be guaranteed that no information leaks on their inputs apart from what is required by the task. The challenge is to design cryptographic protocols for such tasks, which will even function if one or several of the parties (up to a certain fraction) are behaving adversarily.

Secure function evaluation is a well-known task involving multiple parties $P_1, \ldots, P_n$, say, each holding a value $x_1, \ldots, x_n$, respectively, for which they like to evaluate the function value $f(x_1, \ldots, x_n)$ for some agreed upon function $f$. The problem of secure function evaluation is to find a protocol for $P_1, \ldots, P_n$ which enables them to jointly compute output value $f(x_1, \ldots, x_n)$ correctly, however in such a way that their respective input values $x_1, \ldots, x_n$ remain secret, except for the information that can be inferred logically from the output value. This should hold even if a certain fraction of parties $P_1, \ldots, P_n$ behaves adversarially.

The theory of secure multiparty computation tells us that protocols for tasks such as secure function evaluation can actually be designed under reasonable conditions. In this report, the state of the art in this field is summarized and various open problems are identified. The overview is divided into a part treating general primitives, theory and constructions (Chapters 2 and 3), and a part treating specific problems and constructions (Chapter 4). The general part focuses on secure function evaluation, whereas the specific protocols in Chapter 4 span a broader range of applications, including, e.g., anonymous credentials and group key exchange.
Chapter 2

Primitives

In this chapter we review four common (mostly cryptographic) primitives used in multiparty protocols, namely broadcast channels, secret sharing, threshold cryptography, and oblivious transfer. These primitives will be used extensively in the protocols in the later chapters.

2.1 Fault-tolerant Broadcast

Many multiparty protocols assume that parties communicate by a broadcast channel, i.e., all parties receive the same set of messages in the same order, and (roughly) at the same time. While that may hold in a small local network, most medium and large size networks provide point to point communication between the parties only. In other settings, such as a wireless network, parties may have a broadcast channel to some other parties, but not be able to reach all parties directly. As most networks can be modeled as asynchronous point-to-point networks (even though that might result in some performance loss due to the necessary communication protocols), this model is the one best researched, though specialized protocols exist that make use of particular network properties, such as synchronous communication.

There is some work on networks that can break into partitions and rejoin at a later point in time. As most multiparty protocols will be unable to cope with partitions, we do not discuss this model in detail here.

Fault-tolerant broadcast protocols fill the gap between the idealized model (i.e., a broadcast channel) and the real world (i.e., affected by failures and/or partial connectivity) by providing some basic properties of broadcast channels on top of a less reliable network. While it may not be possible to completely simulate a broadcast channel—for example, it is impossible to achieve any timing guarantee using an asynchronous network—primitives exist that can achieve most of the necessary properties.

An asynchronous consistent broadcast primitive guarantees that every party that does receive a particular broadcast receives the same message. While it is not guaranteed that all parties do receive the message if the sender is dishonest, the parties get a guarantee of consistency. The first consistent broadcast protocol in the literature is in [218]. While it is rather efficient in terms of communication, it relies on public key cryptography, which may cause a significant slowdown in some cases.

A reliable broadcast goes one step further than a consistent broadcast and furthermore guarantees that either all honest participants receive the broadcast message, or none does; this property is also called agreement. The first protocol to achieve that goal in an asynchronous
point to point network goes back to Bracha [36]. While this protocol is rather expensive in terms of communication, requiring $O(n^2)$ messages for a broadcast to $n$ parties, it does not require any expensive cryptography.

Finally, an atomic broadcast not only ensures that all parties receive the same set of messages, but furthermore guarantees that they receive all messages in the same order. This property is the hardest to achieve, as one can show that it is equivalent to solving the consensus problem, which itself has been shown to be impossible to solve in a fully asynchronous network [124] (though it is possible to circumvent this result using randomization). However, as the guarantees given by an atomic broadcast are needed for many protocols, a rich literature on implementations exist, using either “somewhat” asynchronous system models [27, 219, 172], or randomization techniques [58]. It has been proposed to increase performance of those protocols using optimistic techniques. Such protocols perform very fast if the network is reasonably well behaved, and may slow down otherwise, but they always maintain safety. The BFT protocol by Castro and Liskov [81] and the KS protocol by Kursawe and Shoup [178] are two examples of such asynchronous atomic broadcast protocols. They have message complexity $O(n^2)$. The BFT protocol is deterministic and may not be live in an asynchronous network, whereas the Kursawe-Shoup protocol is based on randomized agreement, which provides liveness except with negligible probability. Recently, Ramasamy and Cachin [217] have proposed a new optimistic protocol that reduces the message complexity to $O(n)$ in the optimistic mode compared to the previous protocols. It is the first protocol with an amortized expected message complexity of $O(n)$ per delivered payload.

2.2 Secret Sharing Schemes

Secret sharing is a fundamental primitive in cryptography; it was introduced in 1979 independently by Blakley [28] and by Shamir [228]. Secret sharing schemes have been extensively studied along the years [235]. In its basic form, a secret sharing scheme is a protocol divided into a Sharing Phase and a Reconstruction Phase. During the Sharing Phase, a dealer distributes a secret among a set of participants by sending in a secure way a piece of information, called share, to each of them. Then, during the Reconstruction Phase, some subsets of participants (qualified subsets), by pooling together their shares, reconstruct the secret; while, other subsets, (forbidden subsets), are not able to reconstruct the secret. Moreover, if the scheme is such that forbidden subsets, by pooling together and processing their shares, do not get any information about the secret, it is said to be perfect. In such a model, dealer and participants are supposed to be honest.

Efficiency and Information Rate. One of the basic issues in the area of secret sharing schemes is that of estimating the information rate of the scheme, that is, the ratio between the size of the secret and that of the largest share given to any participant. Although the issue has received considerable attention (e.g., [16, 51, 80, 105] to name a few) some interesting problems remain still open. Schemes where the information rate is equal to one are said to be ideal.

Linear Secret Sharing Schemes. An important class of secret sharing schemes is the class of linear secret sharing schemes. Brickell [50] was the first author who considered (ideal) linear schemes and introduced the vector space construction, generalized later by Van Dijk...
General linear secret sharing schemes were studied by Simmons [232]. Geometric secret sharing schemes, considered by Jackson and Martin [160], are also general linear secret sharing schemes. Karchmer and Wigderson [167] showed that monotone span programs realize any linear secret sharing scheme.

**Verifiable Secret Sharing Schemes.** In order to design secret sharing schemes that keep working in hostile environments, the concept of *verifiability* was introduced in [88]. With this more general approach, some extra information is used to enable participants to detect a dishonest dealer, who sends inconsistent shares during the Sharing Phase, and to verify during the Reconstruction Phase that each participant sends a correct share. There is an extensive literature on verifiable secret sharing schemes (e.g., [146, 83, 216, 91, 139] for the unconditionally secure setting and, e.g., [123, 205] for the computationally secure setting). Verifiable secret sharing schemes play a key role in multi-party computation.

**Proactive Verifiable Secret Sharing Schemes.** The concept of *proactive security* was introduced in [201] and applied to the secret sharing setting in [153]. Basically the idea is that, if the shares held by the participants stay the same for all the lifetime of the system, then an adversary can eventually corrupt a sufficient number of participants to learn the secret. On the other hand, if time is divided into *periods*, and at the beginning of each period the stored shares change (while the shared secret stays the same), then the adversary probably does not have enough time to corrupt the necessary number of participants. Moreover, the shares he learns during period $p$ are useless during period $p + i$, for $i = 1, 2, \ldots$. So, he has to start a new attack from scratch during each time period. See [73] for an overview of schemes dealing with the synchronous setting and [57] for the asynchronous one.

**Strongly Multiplicative Hierarchical Threshold Secret Sharing** Strongly multiplicative secret sharing schemes are used in multi-party computation to obtain error-free multiplication unconditionally secure against an active adversary. However, enforcing the multiplication property is in general expensive and few efficient non-threshold examples are known. Käasper and Nikova have proposed in [118] two different solutions for obtaining strong multiplication in the hierarchical threshold setting. The first constructions achieve robustness against the strongest possible adversary. These schemes are not ideal but have a reasonable information rate for hierarchies with few levels. The proposed ideal constructions are strongly multiplicative under somewhat stronger, but still feasible assumptions. In particular, they have proposed a modification that improves the multiplication properties of the scheme. Their results are not tight and it is possible that better bounds can be obtained by more careful analysis. However, the modified scheme has been shown to be optimal with respect to the most crucial property—the number of required top-level participants.

The modified ideal scheme has a randomized identity allocation strategy with failure probability $p = \Theta(1/q)$, which is a safe bet for large field sizes $q$. Still, for the original conjunctive and disjunctive constructions, the authors also proposed a deterministic allocation strategy, which has zero failure probability if the field is sufficiently large. The strategy allocates identities in a monotone fashion, so participants from higher levels get “smaller” field elements. It would be interesting to verify if the deterministic strategy also applies for the new scheme.
Visual Cryptography. A visual cryptography scheme [196] is a secret sharing scheme to encode a secret image $SI$ in such a way that any qualified subset of participants can “visually” recover the secret image, while forbidden subsets have no information on $SI$. A “visual” recovery consists of xeroxing the shares, which are shadow images, onto transparencies and stacking them one on the top of the others. The participants in a qualified subset will be able to see the secret image without any knowledge of cryptography and without performing any cryptographic computation. See [33] for references on recent works on the subject. Visual cryptography is a powerful tool for teaching cryptography to general audience. Applications have also been proposed to realize authentication, identification schemes and, recently, e-voting schemes.

2.3 Threshold Cryptography

Threshold cryptography allows to distribute a cryptographic operation among a group of $n$ parties in a fault-tolerant way. An operation can be carried out despite the failure or malicious behavior of up to $t$ parties and in such a way that no cryptographic secrets are leaked and the result is correct. The secret key of the cryptosystem is shared among the group using secret sharing (see Section 2.2).

Threshold cryptographic extensions exist for most cryptosystems, but the ones that have received most attention so far were public-key encryption and digital signature schemes.

Threshold cryptosystems are distributed cryptographic protocols and differ in the network model and timing assumptions used (synchronous, asynchronous), in the corruption capabilities of the adversary (static, adaptive), and in model used in the cryptographic security proof (random oracle, standard model).

A threshold cryptosystem can either be initialized by a trusted party (usually called the *dealer*), who generates the secret key and distributes its shares among the group, or it can be initialized by a distributed protocol that generates the secret key using a threshold cryptographic protocol. The latter approach is necessary in order to completely eliminate any single points of failure, or trust.

In this section we consider threshold encryption and threshold signatures for RSA-based systems, discrete logarithm-based systems, and for systems based on Paillier’s composite-degree residuosity classes.

There are many extensions of threshold cryptography. The most important among them are proactive cryptosystems [152], which introduce a periodic refresh protocol for refreshing the shares so that more than $t$ faulty parties can be tolerated over the lifetime of the secret key, provided that no more than $t$ are faulty between two executions of the refresh protocol. Proactive cryptosystems are usually obtained by replacing the secret sharing scheme of a threshold cryptosystem with a proactive verifiable secret sharing scheme (see Section 2.2), but sometimes additional work is necessary [213].

2.3.1 Models

Network Model. The network typically provides at least pairwise authenticated point-to-point channels among all $n$ parties in the group; sometimes the channels are also assumed to be private. Often threshold cryptosystems are designed on the basis of assuming a broadcast channel. A broadcast channel is an abstraction which guarantees that all non-faulty parties
receive the same set of messages through the channel and that they receive them in the same order. It can be implemented using a protocol for Byzantine agreement, see also Section 2.1.

The network can be synchronous or asynchronous. In a synchronous network, there are known upper bounds on the delay of messages on any channel and on the relative clock speeds of the parties. One can imagine that all communication in synchronous network occurs in a sequence of global rounds using either the point-to-point channels or a broadcast channel. Most threshold cryptographic protocols assume this idealized model.

In an asynchronous network, no common clock exists and the delay of all messages on the point-to-point channels is unbounded. Typically an adversarial scheduler is assumed to control the delivery of messages over the network. There are only a few asynchronous threshold cryptographic protocols.

Non-interactive threshold cryptosystems proceed without interaction among the parties and typically require only one computation step for every party, in which it generates its share of the cryptographic result (such as a “signature share” or a “decryption share”); enough correct shares can then be assembled to the result by anyone. Such schemes are very practical and also suitable for asynchronous networks.

Corruption Model. One distinguishes between static and adaptive corruptions in cryptography: in the static corruption model, the adversary must decide whom to corrupt independently of the execution of the system, whereas in the adaptive corruption model, the adversary can adaptively choose whom to corrupt as the attack is ongoing, based on information it has accumulated so far. It is strictly easier to design protocols for the static case than for the adaptive case.

Cryptographic Proof Model. As other cryptographic schemes, threshold cryptosystems can be proved secure either in the standard model or in the so-called random oracle model. The latter falls short of a proof in the standard model, but nevertheless gives very strong evidence for the security of a construction. Schemes in the random oracle model are attractive because they are practical and require less interaction; non-interactive threshold cryptosystems commonly use the random oracle model.

Threshold Cryptosystems. A threshold public-key cryptosystem looks similar to an ordinary public-key cryptosystem with distributed decryption. There is a single public key for encryption, but each party holds a key share for decryption. The decryption operation is usually an interactive protocol among the parties. In non-interactive schemes, however, the parties process decryption requests for a particular ciphertext and simply output a decryption share together with a proof of its validity. In an interactive protocol with input a ciphertext resulting from encrypting some message and given that at least $t + 1$ correct parties engage in decryption, it must be easy to recover the message; similarly, in the non-interactive setting, given more than $t$ valid decryption shares for that ciphertext, it is easy to recover the message. This property is called robustness. Threshold cryptosystems should also be secure against adaptive chosen-ciphertext attacks in order to be useful for all conceivable applications.

Very efficient discrete logarithm-based threshold cryptosystems are known. Shoup and Gennaro [231] present a non-interactive ElGamal-like system that is secure against adaptive chosen-ciphertext attacks in the random oracle model. A less efficient protocol that requires
a synchronous network with broadcast was presented by Canetti and Goldwasser [76]; it is based on the Cramer-Shoup cryptosystem and assumes only the standard model. Earlier constructions did not provide security against adaptive chosen-ciphertext attacks.

For RSA, there are practical non-interactive protocols for carrying out the basic decryption operation, which are described below in the section on signature schemes. But they only give rise to cryptosystems that are secure against chosen-plaintext attacks; designing a non-interactive RSA-based threshold cryptosystem that is secure against adaptive chosen-ciphertext attacks is an interesting open problem.

**Open Problem No. 1:** Design a non-interactive RSA-based threshold cryptosystem that is secure against adaptive chosen-ciphertext attacks.

It is also known how to construct threshold versions of Paillier’s cryptosystem that are secure against chosen-ciphertext attacks in the standard model with synchronous broadcast, as shown by Fouque and Pointcheval [127].

### 2.3.2 Threshold Signature Schemes

In a *threshold signature scheme*, each party holds a *share* of the secret signing key. In interactive schemes, the parties carry out a distributed protocol to generate a signature for a message given as input. In non-interactive schemes, every party generates shares of signatures on individual messages upon request, and the validity of a signature share can be verified for each party. From \( t + 1 \) valid signature shares, one can generate a digital signature on the message that can later be verified using the single, publicly known signature verification key. In a threshold signature scheme with security in the sense of existential forgery against adaptive chosen-message attack, it must be infeasible for a computationally bounded adversary to compute a valid signature on a message that no honest party took as input to the signing protocol or for which no honest party generated a signature share (in non-interactive schemes).

Threshold signatures based on the hardness of the discrete logarithm were proposed for the DSA in synchronous networks with broadcast [140]. They can also be extended to tolerate an adaptive adversary [74, 129].

For RSA signatures, synchronous protocols in a network with broadcast are known in the standard model [144, 215]. A very efficient, non-interactive solution in the random oracle model was presented by Shoup [230], although the choice of RSA modulus was restricted to the product of two safe primes in this work. Fouque and Stern [128] and Damgård and Koprowski [108] show how to remove that restriction.

### 2.3.3 Threshold Pseudorandomness

A threshold *pseudorandom function*, also called a threshold *coin*, is a protocol to evaluate a pseudorandom function whose secret key is shared among the group. *Asynchronous* and *non-interactive* threshold pseudorandomness schemes can also be used to implement asynchronous protocol for Byzantine agreement efficiently [59, 198]. Several interactive and non-interactive protocols of this kind are known in the standard model [195, 198] and in the random oracle model [194, 59].

**Open Problem No. 2:** Are there practical non-interactive threshold random functions in the standard model (not using random oracles)?
In contrast to a pseudorandom function, a pseudorandom permutation can directly be used for symmetric-key encryption, i.e., it provides the abstraction of a block cipher such as AES. It is therefore a very widely used primitive in practice. A distributed implementation, for example, could be used to distribute a key-distribution center as used in Kerberos. Yet, no efficient threshold pseudorandom functions were known so far. Only very recently, Dodis et al. [116] constructed a threshold pseudorandom permutation with “reasonable” efficiency, based on the well-known Luby-Rackoff construction. Its main feature is to use only a constant number of rounds for evaluation, but the necessary computations render it still far from being practical.

**Open Problem No. 3:** Provide a practical, perhaps even non-interactive, threshold implementation of a pseudorandom permutation.

### 2.3.4 Threshold Key Generation

Generating a threshold key for the discrete log-based systems is easier than for RSA or other trapdoor one-way permutations. Pedersen’s verifiable secret sharing scheme [205] gives rise to a protocol for generating shares of an ElGamal public key, but it requires an unrealistically strong synchronized network with broadcast. Gennaro et al. [141] show how to generate a shared threshold key with a weaker and more realistic synchrony assumption, tolerating a so-called rushing adversary that can exploit small communication delays to its advantage. Fully asynchronous protocols for generating discrete log-based threshold keys were proposed only recently by Cachin et al. [57].

The distributed generation of RSA threshold keys was addressed in synchronous networks by Boneh and Franklin [32]; these results were later also extended to generate keys for Shoup’s RSA scheme [3, 108]. No asynchronous protocols are known.

**Open Problem No. 4:** Design practical and robust protocols for distributed generation of shared RSA keys in a variety of communication models.

### 2.4 Oblivious Transfer

Oblivious Transfer is a two-party protocol involving a sender and a receiver and lies at the core of secure multiparty computation protocols. In an oblivious transfer, the sender possesses \( t \) secret strings of \( k \) bits each, and agrees to provide the receiver with one of these strings. The security requirements are that the sender does not figure out which secret the receiver has obtained and that the receiver does not learn anything about the other \( t - 1 \) strings of the sender. We will denote a Oblivious Transfer among 2 participants where the receiver obtains 1 secret out of \( t \) secrets of \( k \) bits by \((t_1) \leftarrow OT_{k}^t\). This form of oblivious transfer is also known as All-or-Nothing Disclosure of Secrets (ANDOS).

Originally, Oblivious Transfer was introduced by [214] for the slightly different case of \((1) \leftarrow OT_{k}^1\), where the sender has only one secret bit. The receiver either obtains this or a uniformly random bit with probability \(1/2\), but does not know which case occurred. Surprisingly, this unusual primitive is extremely powerful and equivalent to the other variations in power. Oblivious transfer quickly became a fundamental building block for secure computation protocols. Kilian [173] showed that any computable function can be evaluated in a secure computation with oblivious transfers.
After the original Rabin article, the case $(\frac{t_1}{1}) - OT_2^k$ was studied in [240], the case $(\frac{t_1}{1}) - OT_1^1$ in [119] and the general case $(\frac{t_1}{1}) - OT_2^k$ in [43] with a more practical approach.

Research on oblivious transfer has been focused on several questions henceforth. One direction has been the attempt to realize oblivious transfer using quantum cryptography [26, 100], but this was later shown to be impossible in the pure quantum model [101].

Another approach investigates reductions among the basic flavors of oblivious transfer, such as reductions from $(\frac{t_1}{1}) - OT_2^k$ to $(\frac{t_1}{1}) - OT_2^k$ [44], or weaker forms of the primitives, such as universal oblivious transfer [42, 54].

It has also been shown how to realize oblivious transfers in different security models or from other primitives, such as in the storage-bounded model [56, 115] or in the noisy channels model [102]. The latter direction has revealed a fundamental equivalence between the certain noisy channels and oblivious transfer [107].

The related problem of private information retrieval (PIR) has also received interest, given the raising interest in privacy protection technologies [87]. In its pure form, PIR considers only the privacy of a user who retrieves an entry from a database. This corresponds to an oblivious transfer with security only for the receiver against the sender. But, as a matter of fact, the security guarantee for the sender has also been introduced in that model, leading to a primitive called Symmetrically Private Information Retrieval (SPIR) corresponding to ANDOS.

Considerable attention has been devoted to the construction of efficient PIR and ANDOS schemes recently [87, 179, 234, 60, 193, 159].

We observe the following open problems related to oblivious transfer.

Open Problem No. 5: Oblivious transfer/ANDOS lies at the heart of secure multiparty computation protocols; since multiparty computation has very recently been demonstrated also in practical prototypes [184], an important problem is to find efficient protocols for ANDOS.

Open Problem No. 6: The known oblivious transfer protocols in the bounded-storage model suffer from the drawback that the storage space required by the users is rather high compared to the maximum storage allowed to the partner so that the protocol still remains secure (the required space is roughly the square root of the bound). It is an open problem either to lower the required amount of storage or to show that no such reduction is possible.

Concurrent oblivious transfer. More recently, research has focused on the use of oblivious transfer protocols in the cryptographic security model as a building block for more complex protocols. In [137], the problem of constructing a protocol for concurrent oblivious transfer has been solved by using a black-box simulator and considering the simpler case that the inputs of all parties are chosen independently in every execution. In [181], it is shown that actually when this restriction is removed then there does not exist a non-trivial protocol for concurrent oblivious transfer using black-box simulation.

Open Problem No. 7: Since non-black-box oblivious transfer is not included in the set of functionalities for which the impossibility results of [181] hold, an important open question is whether or not non-black-box simulation can be used to achieve concurrent oblivious transfer in the plain model.
Universally composable oblivious transfer was studied in [138]. An efficient construction in the multi-party setting with honest majority is discussed in [125].

**Adaptive oblivious transfer.** In an $m$-out-of-$n$ adaptive oblivious transfer protocol, denoted $(^n_m) - OT^a_2$, the receiver can dynamically choose up to $m$ messages from a list of $n$ messages offered by the sender. This can be done in a generic way by letting the sender commit to the messages and performing $m$ executions of an $(^n_1) - OT^a_2$ protocol on the opening information, but more efficient direct protocols exist as well. Naor and Pinkas [193] propose one requiring $O(\log n)$ invocations of a $(^n_1) - OT^a_2$ protocol, resulting in a communication overhead of $O(\log n)$ during each transfer. In the random oracle model, direct constructions with constant communication overhead in the transfer phase exist based on RSA [200] and pairings [89]. These were later found to be special cases of a generic construction from unique blind signatures [67]. A direct protocol with constant communication overhead in the standard model was also presented in [67]. Schemes with logarithmic overhead but relying on weaker security assumptions were proposed in [148].

**Open Problem No. 8:** Is it possible to obtain adaptive oblivious transfer protocols with constant communication overhead in the transfer protocol (i.e., independent of the total number of messages $n$) under “standard” security assumptions? The protocols of [67] rely either on interactive “one-more” assumptions in the random oracle model, or on non-standard pairing assumptions. The protocols of [148] have communication complexity $O(\log n)$ in the transfer phase.

A slightly more flexible variant of adaptive oblivious transfer is *oblivious keyword search*. Here, the messages offered by the sender are not indexed through sequence numbers, but through arbitrary bit strings called *keywords*. The receiver can query the message associated to a keyword of his choice. Efficient constructions can be obtained from unique blind signatures [200, 89, 67], from oblivious polynomial evaluation [134], and from oblivious pseudorandom functions [134].

**Oblivious Transfer and Commitments.** Commitments can be used in several ways to strengthen the security properties of oblivious transfer protocols. For instance, as mentioned above, commitments are used as a building block in adaptive oblivious transfer. In general, combinations of commitments and oblivious transfer can be distinguished based on (a) when the commitments are available, and (b) whether bits or bit-strings are transferred. Below, we distinguish three types of combinations.

Committed Oblivious Transfer (COT) is a variation where commitments to the inputs have been established prior to the protocol execution, and at the end of the execution there is a commitment by the receiver to its private output. The first protocol for COT is presented by Crépeau [99] for the case where one out of two bits is being transferred, and later improved by Crépeau et al. in [104], both achieving stand-alone security. These results are improved further by Garay et al. [138] in terms of efficiency, and, as mentioned above as well, using the universally composable framework of [69] in the CRS model. In all these cases, it is shown how to build general secure multiparty computation protocols on top of COT, in the presence of malicious adversaries.

Transferring bit-strings in COT can be achieved by performing parallel invocations to COT for bits, treating each bit separately. A more efficient protocol is presented by Kiraz et
al. [176] based on any (2,2)-threshold homomorphic cryptosystem set up between the parties wishing to transfer bits. This protocol transfers the bits in one go and is secure in a stand-alone manner, requiring essentially the same computational and communication effort as COT of [138] for one bit.

Open Problem No. 9: Is it possible to obtain Committed Oblivious Transfer in the UC-framework dealing with adaptive adversaries in the non-erasing model?

Verifiable Oblivious Transfer (VOT) is a variation of COT in which no commitment to the receiver’s output is generated by the protocol. Commitments to the inputs, both of the sender and the receiver, are required though. Cachin and Camenisch [55] present a VOT protocol for transferring bits, while Jarecki and Shmatikov [162] present a VOT protocol for transferring bit-strings in the UC framework. Both protocols are also applied to get general secure multiparty computation.

As a final variation, we mention Committing Oblivious Transfer, where the sender and receiver start as in a plain oblivious transfer (without any commitments for their inputs). Upon completion of a Committing Oblivious Transfer protocol, however, the sender and receiver are committed to (some of) the actual inputs used by them in the protocol run. Possibly, the receiver is committed to its output as well. Committing OT protocols have been shown to be sufficient to eliminate certain flaws which arise from the use of plain OT in some protocols for two-party computation based on Yao’s garbled circuit construction [175]. The flaws arise from the use of plain OT as a subprotocol, leaving the possibility for a malicious sender to use bogus values as input to OT, in order to compromise privacy for the surrounding protocol. Committing OT eliminates such issues by allowing additional verification based on the commitments produced. An example OT protocol that fits with committing OT ideas is presented by Lipmaa in [183]. That protocol is presented under the name Verifiable Homomorphic Oblivious Transfer, where homomorphic means that it is based on homomorphic encryption and commitment schemes, and verifiability is defined in the sense that at the end of the protocol there will be commitments to all inputs of OT. The idea is that those commitments are used or referred to by any surrounding protocol afterwards, allowing, for example, the verification of the actual inputs used in the OT protocol.
Chapter 3

Generic protocols

This chapter is mostly devoted to a discussion of the generic approach to multiparty computation (MPC). The generic approach allows for securely computing any function representable as a polynomial-sized circuit. Before treating the multiparty case, however, the special case of two-party computation is briefly separately in the first section of this chapter.

3.1 General Secure Two-Party Computation

A two-party \((P_0, P_1)\) protocol problem is described by specifying a random process that maps pairs of inputs to pairs of outputs by means of a functionality. Specifically, \(f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*\), where \(f = (f_0, f_1)\), is the functionality such that given a pair of inputs \((x_0, x_1)\), the output is a random variable \((f_0(x_0, x_1), f_1(x_0, x_1))\). \(P_0\) runs on input \(x_0\) and obtains \(f_0(x_0, x_1)\) while \(P_1\) runs on input \(x_1\) and obtains \(f_1(x_0, x_1)\). From the current state-of-the-art, we know how to securely implement (in the computational setting) any two-party protocol assuming the existence of trapdoor permutations [243, 146].

In the more realistic setting in which different executions are concurrently composed, each party has one input for each execution. Unfortunately, a protocol that is secure in the stand-alone setting is not necessarily secure under concurrent composition. As previously discussed, in the strong notion of universal composability [69] security holds under concurrent general composition. Moreover, in the case of a honest majority, universally composable protocols exist in the plain model for any functionality. Unfortunately, in case of dishonest majority, in the plain model there are many functionalities that cannot be securely computed in a universally composable sense [77].

For the case of concurrent self-composition, very important feasibility results in the plain model have been recently achieved (see [180, 204, 203]) for the case of bounded concurrency. Surprisingly for many functionalities, security under concurrent self composition implies security under concurrent general composition [181].

It is important to study the alternative models in which protocols can be composed since this is what really happens in the real world. In particular, many models have been proposed for achieving results on zero-knowledge protocols that instead are impossible in the plain model for non-trivial languages [75, 117]. In such models, feasibility results on secure two-party computation are still possible.

In the common reference string model, universally composable two-party and multi-party computation has been achieved in [78], where also the corresponding results for the shared
random string model are described.

When an a-priori bound on the maximal number \( m \) of concurrent executions is assumed and if the protocol is proven secure using black-box simulation, then more than \( m \) rounds of communication are needed and any two-party functionality that has to be securely computed under \( m \)-bounded concurrent self composition [180].

In the broadcast channel model constant-round secure multi-party (and thus two-party) computation with a dishonest majority has been achieved in [168]. The result has been improved in [203] in the plain model (without requiring a broadcast channel) achieving also bounded concurrency. However, it is still important the possibility of achieving a stronger notion of composition by means of a broadcast channel.

Another important open question is the possibility of relaxing the strong notion of universal composability, still preserving a satisfying security. One step in this direction has been performed in [212].

Solved Problem No. 1 (solves Open Problem No. 2 from [206]): Recently Kalai et al. [166] and Barak and Sahai [12] have shown how to construct protocols for concurrent general composition with timing assumptions and under superpolynomial-time simulation.

### 3.2 The Generic MPC Problem

We now turn from the two-party case to the multiparty case.

The generic MPC problem is to devise protocols, which allow \( n \) parties connected by a complete network of point-to-point channels to securely compute any function \( f \) on their local inputs and to study under which assumptions and how efficiently such protocols can be devised. Securely compute means preserving the correctness of the outputs and the privacy of the inputs in the presence of faults. Faults are modeled by a single entity called the adversary. The adversary controls a set of so-called corrupted parties. If the adversary is passive it sees all inputs and outputs of the corrupted parties and can view their internal state. An active adversary is furthermore allowed to send arbitrary messages on behalf of the corrupted parties. An adversary has two goals, gathering information about uncorrupted parties' inputs and outputs and, if it is active, exerting influence by sending faulty messages to make the uncorrupted parties terminate with wrong outputs.

A solution to the generic MPC problem is a compiler which given the description of any protocol problem, formalized by a function \( f \), outputs a secure protocol for that protocol problem.

### 3.3 Formalizing the Problem

There are many proposals on how to model the security of a protocol, but common to most is that the security of a protocol is defined by requiring that an adversary attacking the protocol should only be allowed to achieve inevitable goals. This meaning that the adversary should not be able to achieve more than it would from attacking an ideal evaluation of \( f \). In the ideal evaluation we imagine that the parties send their inputs to an incorruptible trusted party over perfectly secure lines. This trusted party then computes the function \( f \) on the inputs and returns the result over the perfectly secure lines. Here certainly the adversary archives
only inevitable goals: Of information it only learns the corrupted parties’ inputs and outputs
and if it is active the only influence it can exert is changing the corrupted parties’ inputs
to the function. The comparison of the protocol execution to the ideal evaluation is made
by requiring that the complete view of an adversary attacking the protocol execution can be
simulated given only the view of a correspondingly powerful adversary attacking the ideal
evaluation. This captures exactly the idea that the information gathering and the influencing
capabilities of the adversary contain nothing extra to that of which the adversary is entitled.
This approach to comparing the protocol execution to the ideal evaluation originates in the
definition of zero-knowledge proof in [147] by Goldwasser, Micali and Rackoff, where the same
approach is used to define that the view of a dishonest verifier does not contain information
about the witness by showing that the view can be simulated without the witness. We call
this kind of model of security a simulation model. For the MPC setting the simulation model
approach is introduced by Goldreich, Micali and Wigderson [146].

There now exist a number of approaches to formalizing this intuitive definition. The
formalizations currently regarded the ‘right’ ones are the universal composability (UC) frame-
work by Canetti [69] and the reactive simulatability (RS) framework by Pfitzmann, Schulter
and Waidner [209]. Both frameworks guarantee security for complex environments as e.g. the
Internet, where earlier formalizations all had drawbacks. A comprehensive overview of earlier
formalizations can be found in [68].

The models used to study MPC vary considerably in the assumptions on the computing
power of the parties, the nature of the communication channels, and the class of faults, which
are considered. Some common choices for these parameters give rise to the cryptographic
models and the information theoretic models. In the cryptographic models, the parties and
the adversary are assumed to be probabilistic polynomial time interactive Turing machines
(ITM) and the channels are assumed to be authenticated but public, i.e. the communication
over the channels is publicly readable.

In the information theoretic models the adversary is allowed infinite computing power,
but then the channels are authenticated and private. I.e., the adversary now cannot read
messages sent between honest parties. Thus research in the information theoretic models so
to say investigates which protocol problems can be solved based solely on the assumption
that secure communication is possible.

3.3.1 The early feasibility results
In [84] Chaum, Damgård, and van de Graaf and independently in [146] Goldreich, Micali,
and Wigderson proved that if less than \( n/2 \) of the parties deviate from the protocol, then any
probabilistic polynomial time \( n \)-party function can be computed securely in the cryptographic
model. In [83] Chaum, Crépeau, and Damgård and independently in [22] Ben-Or, Goldwasser,
and Wigderson proved, that if less than \( n/3 \) of the parties are cheating, then any \( n \)-party
function can be computed securely in the information theoretic model. In [216] Rabin and
Ben-Or proved that in the information theoretic model with a broadcast channel any \( n \)-party
function can be computed securely as long as less than \( n/2 \) parties are actively cheating.
These early protocols are not particularly efficient, but showed that secure generic MPC is
feasible, and set the stages for a now very active research area.
3.4 The Typical Setting

When studying MPC there are three assumptions which are typically made. These are: the computation is synchronous, the network is complete, and the channels are authenticated.

The reason for considering a synchronous model is that it simplifies the study without (hopefully) abstracting away the problems that are essential to secure MPC. The hope is, that tools and methodologies developed in the synchronous model for secure computation can be adapted to the asynchronous model. There is good evidence, that this is the case. If we do not consider active faults, then any privacy preserving protocol for the synchronous model can be translated into a privacy preserving protocol for the asynchronous model using a synchronizer, see Awerbuch[9]. Even though no such construction is known for the active case, many of the tools developed for the synchronous model have been adopted to protocol design in the asynchronous setting. The concept of verifiable secret sharing (VSS) was defined for the synchronous model by Chor, Goldwasser, Micali and Wigderson [88], where it was used to construct a so-called simultaneous broadcast protocol. VSS is an essential part of most MPC protocols. In [79] Canetti and Rabin adapted VSS to the synchronous model. They build an asynchronous VSS (AVSS) protocol and use it to construct asynchronous Byzantine agreement. Also Ben-Or, Canetti and Goldreich [20] and Ben-Or, Kelmer and Rabin[23] used AVSS, as a sub-protocol in building generic MPC protocols for the asynchronous information theoretic model. The model introduced in [20] builds directly on the definitional work for the synchronous model begun in [145] and both protocols use the tools and methodologies from the protocols for synchronous MPC protocols in [83, 22, 216].

The motivation for considering complete and authenticated networks is partly the same as for studying synchronous computation. For these assumptions, the hope that results in the idealized model apply to more realistic models is even better justified. In [17] Bellare, Canetti and Krawczyk showed how any protocol for the authenticated channel model can be translated into a protocol for the unauthenticated channel model in a modular way using known (and widely applied) techniques for authentication, like authenticated Diffie-Hellman for key exchange and messages authentication codes (MAC’s) and digital signatures for data transport. A further discussion of this issue and references to related work can be found in [17].

A study of which general networks allow for simulating a complete network, and performing MPC, has been performed in a number of papers. E.g. in the information theoretic model, if the network consists of point-to-point channels and $t$ parties might be corrupt, but do not actively cheat, then $t + 1$ disjoint paths between two parties are necessary and sufficient to simulate a complete point-to-point channel. If the $t$ corrupted parties may actively cheat, then $2t + 1$ such paths are necessary and sufficient, see e.g. Franklin and Wright[132], which also contains similar result for other communication models with e.g. multicast and broadcast.

It is the hope that studying secure computation in an idealized model will develop efficient tools and methodologies which have wider applications to protocol design and that it will clarify what is essential aspects of (efficient) secure computation. Furthermore, secure MPC is a place for benchmarking existing tools and methodologies for protocol design.
3.5 State of the Art and Open Problems

In this section we take a look at the state of the art of generic MPC and describe some open problems. When looking at the state of the art, our focus will be on efficiency, in particular the communication complexity of protocols, measured in bits. There are other efficiency measures which could be considered, like the communication complexity in number of messages, the communication complexity in rounds and the time complexity of the computations done by the protocol. The reason for focusing on the bit complexity is that the bit complexity seems to be the main practical bottle-neck responsible for current generic MPC protocols not being practical.

The MPC problem dates back to the papers by Yao [244] and Goldreich, Micali and Wigderson [146]. What was proved there was basically that a collection of \( n \) players can efficiently compute the value of an \( n \)-input function, such that everyone learns the correct result, but no other new information. These protocols are proved secure against static PPT adversaries which corrupts a set of less than \( n/2 \) players. Both protocol are for the cryptographic model and are proved secure in models which are somewhat weaker than the UC model and the RS model. Later, unconditionally secure MPC protocols were proposed by Ben-Or, Goldwasser and Wigderson and Chaum, Crépeau and Damgård [22, 83] in for the information-theoretic model where private channels are assumed between every pair of players.

3.5.1 The VSS Strain

Over the years, several protocols have been proposed which, under specific computational assumptions, improve the efficiency of general MPC, see for instance [11, 93, 143, 91]. Virtually all proposals have been based on some form of verifiable secret sharing (VSS), i.e., a protocol allowing a dealer to securely distribute a secret value \( s \) among the players, but where both the dealer and/or some of the players may be cheating and must still guarantee that after the sharing protocol terminates, a unique secret is defined and can be reconstructed (see also Section 2.2). Then the parties manipulate these shared inputs to compute some function of them. The basic paradigm that has been used is to ensure that all inputs and intermediate values in the computation are VSS’ed, since this prevents the adversary from causing the protocol to terminate early or with incorrect results. This line of work of using VSS for computation has received an immense attention. See e.g. [142] for a starting point.

In all these protocols, the total number of bits broadcast was at least \( n^2k|C| \), where \( n \) is the number of players, \( k \) is a security parameter, and \( |C| \) is the size of a circuit computing the desired function. Here, \( C \) may be a Boolean circuit, or an arithmetic circuit over a finite field, depending on the protocol. Using the best broadcast protocols these protocols would have a communication complexity of at least \( n^4k|C| \).

Hirt, Maurer and Przydatek [155, 154] present the currently most efficient secure general multiparty computation protocol for the information theoretic model, using the so-called player-elimination technique. This protocol has a communication complexity in the order of \( n^2|C| \) bits.

3.5.2 The Threshold-Encryption Strain

In [130] Franklin and Haber proposed a new approach to the MPC problem. Their protocol is not based on secret-sharing, but instead on a so-called joint encryption scheme, where a
ciphertext can only be decrypted with the help of all players, but still the length of an encryption is independent of the number of players. In the protocol all parties broadcast encryptions of their inputs. Then the parties manipulate the encryptions to obtain an encryption of the output. Finally the parties jointly decrypt just the encryption of the output, thereby revealing the output and keeping inputs and intermediary values secret.

The protocol in [130] is only secure against passive adversaries, but is the first protocol to achieve communication complexity in the order of $n^2 k|C|$. Later Cramer, Damgård and Nielsen [95] gave a protocol along the same lines, now with security against static adversaries. This protocol is based on a threshold cryptosystem (see Section 2.3) and has broadcast complexity in the order of $n^3 k|C|$. Recent work by Hirt and Nielsen [156] has reduced this communication complexity to $n^2 k|C|$. This means that when it comes to communication complexity of MPC, the state of the art is that one has to communicate at least $n^2$ bits per gate in the circuit for computing $f$. On the other hand we have no evidence that this is necessary. If the parties need to agree on the outputs, then we know that they must send at least $n^2$ bits in the output phase — we don’t even know whether they need to sends $n^2$ bits for each output bit. For the intermediary values in the gate we have no evidence that the parties even have to communicate. This poses one of the main open problems in generic MPC.

**Open Problem No. 10:** Investigate the communication complexity of secure MPC, by giving more efficient protocols and by proving lower bounds.

### 3.5.3 Adaptive Security

An interesting sub-problem in generic MPC is that of obtaining security against an adaptive adversary which can corrupt the parties at any point in the execution, as opposed to static adversaries which picks the set of corrupted parties before the protocol execution. Most of the protocols mentioned above is only secure against static adversaries.

That a protocol is secure against adversaries which might corrupt parties during the computation does not follow from the static security of the protocol. In general, constructing adaptive secure protocols is considerably more involved than constructing static secure ones. One example of this is the difference between so-called semantic secure encryption and non-committing encryption. Semantic secure encryption is the notion of security normally used today when considering encryption (c.f [18]). It is however known that in many protocols semantic secure encryption is not sufficient for obtaining adaptive security. Instead a so-called non-committing encryption scheme is needed. Such encryption schemes have been developed by e.g. Beaver and Haber [15] and Canetti, Feige, Goldreich and Naor [71], but are more involved and less efficient that semantic secure encryption.

Another example that adaptive security is harder than static security is given by Cramer, Damgård, Dziembowski, Hirt and Rabin [92]. They give an example of a protocol which is secure against computationally unbounded adversaries, but is insecure against PPT adaptive adversaries. Yet another example is provided by Beaver [14] who showed that the protocol in [146], which was proved static secure, is not adaptive secure, unless factoring is intractable or the polynomial time hierarchy collapses.

There does however exist adaptive secure generic MPC. Indeed, the very first general MPC protocols in [22, 83] for the secure channels model are adaptive secure if the secure channels (which the protocols assume) is implemented using non-committing encryption. Later
Canetti, Lindell, Ostrovsky and Sahai [78] and Damgård and Nielsen [109, 110] developed adaptive secure protocols which are additionally secure on the UC and RS models (i.e. they have security appropriate for e.g. a setting like the Internet).

As mentioned above it is known that non-committing encryption is needed in some protocols, and some protocols have been proved to not be adaptive secure. Unfortunately the reasons for which we know that non-committing is needed, and the reason for [146] not being adaptive secure, are rather technical. Non-committing encryption seems to be needed because of the way our models are formulated rather than because of practical security issues. The same applies to the protocol in [146], which does not seem to allow any practical attack by an adaptive adversary. Since non-committing encryption, and adaptive security in general, is sometimes much less efficient than semantic secure encryption, this poses the question of whether e.g. non-committing encryption is really needed for practical security?

**Open Problem No. 11:** Better understand adaptive security, by e.g. giving an example where implementing secure channels with ordinary semantic secure encryption instead of non-committing encryption leads to a practical security problem, or by developing new models of adaptive security which allow to implement secure channels using ordinary semantic secure encryption and maybe renders the protocol in [146] adaptive secure.

### 3.5.4 Asynchronous Security

Another interesting problem in generic MPC is constructing protocol secure when run in an asynchronous network. For the VSS strain the first secure asynchronous protocols was developed by Ben-Or, Canetti and Goldreich [20] and Ben-Or, Kelmer and Rabin [23]. The line of work on making the VSS strain efficient does however not apply to these protocols, and as a consequence the best known asynchronous VSS based generic MPC protocols are very far from the best synchronous protocol in terms of efficiency.

**Open Problem No. 12:** Is it possible to adapt the techniques used to make the synchronous VSS strain of protocols efficient to the asynchronous VSS strain of protocols?

Recently Hirt, Nielsen and Przydatek [157] gave a secure asynchronous protocol based on threshold encryption. The communication complexity of this protocol is in the order of $n^3k|C|$. This means that also for the threshold-encryption strain of generic MPC there is a noticeable gap between the best synchronous and the best asynchronous protocol.

**Open Problem No. 13:** Investigate whether secure asynchronous protocols are inherently less efficient than synchronous protocols, by either giving lower bounds on the communication complexity of asynchronous MPC or developing more efficient protocols.

#### 3.6 Aside: Strongly Multiplicative Linear Secret Sharing

Most generic protocols for MPC are based on Shamir’s secret sharing scheme [228]. A generalization of this is a class of schemes known as Linear Secret Sharing Schemes (LSSSs),
see Section 2.2, which enable MPC in scenarios that are more general that what is otherwise possible, e.g., cases where we do not place the same amount of trust in all players.

In order to enable general computations, the LSSS used must have an extra property: we say it must be multiplicative. Not all LSSSs are multiplicative, but it can be shown [94] that one can, from any LSSS, construct a new one that is multiplicative, and where shares of secrets are at most twice as large.

This leads to protocols with zero error probability secure against passive attacks, i.e., where players are honest but curious. It also leads to protocols secure against active attacks, i.e., where corrupt players do not follow the protocol. Here, however, we can only guarantee a small, but non-zero probability of error.

To get an error-free protocol in the active case would require an LSSS with a further property: it would have to be strongly multiplicative. It is not known whether a result holds for strong multiplicativeness, similar to the one above. This is a major open problem in linear secret sharing.

**Open Problem No. 14:** Can one construct from any LSSS a strongly multiplicative LSSS that is almost as efficient?
Chapter 4

Specific Applications and Multiparty Computation Protocols

The previous chapters presented the most important basic tools and generic constructions for multiparty protocols. This chapter explores some of their application domains. Several practical scenarios are addressed by presenting specific protocols, such as (i) fair exchange, (ii) private matching, (iii) anonymity-supporting credential systems, (iv) property-based attestation, (v) group key exchange, (vi) applications to economic mechanisms such as auctions, and (vi) electronic voting; these are the subject of Sections 4.1–4.7, respectively.

4.1 Fair exchange

The goal of a fair exchange protocol is to exchange digital items between two or more parties, such that at the end either all parties received the item, or none did. Examples are the exchange of digital signatures (contract signing), electronically purchasing an item, or certified email. In most instances of this problem, the items exchanged are either signatures on a common document, or a non-reputable receipt against a document. A restriction to those cases is helpful, as this allows one to verify the correctness of the exchanged items before the exchange is completed.

There is no straightforward way to solve this problem. The reason is that in every protocol, some party must send a last message – and as this party is not going to receive any message anymore at this time, it must already have received its item. Thus, when it omits the last message, which usually leads to the other party not completing the exchange, the protocol is unfair.

In gradual-release protocols, the partners exchange the items in small pieces (bit by bit, or even in fractions of a bit). In every step, the computational effort to reconstruct the item without help of the other party becomes a bit smaller – thus, if one party decides to interrupt the protocol, it has only a limited computational advantage over the other party. The disadvantage of this protocol, apart from a relatively poor performance, is that it assumes both parties to have comparable computing power, which in many cases cannot be assumed. Furthermore, if the items to be exchanged are of sufficient value (such as a contract for selling a house, see [208]), one party might be motivated to donate years of computing to finish the computation, leaving the other party in an undefined state – can the seller of a house look for a new buyer, or will the old one turn up in 5 years with the completed contract?
Another approach is the use of probabilistic protocols [21, 187]. These methods have a randomized running communication protocol so that an attacker does not know when the protocol will end. Thus, while the protocol still can be attacked by omitting the last (meaningful) message, the attacker does not know which message is the last until after he sends it. This approach solves some of the above problems, but remains both inefficient and insecure – the probability of failure decreases only linearly in the number of protocol rounds, i.e., even allowing for a very long runtime does not result in a failure probability which is acceptable by cryptographic standards.

To achieve full fairness, the existence of a trusted third party (TTP) can be shown to be necessary [120]. Initial protocols assumed an online TTP, i.e., a TTP that is always part of the protocol. While protocols exist that can deal with semi-honest parties (i.e., a trusted third party that tries to learn what items are exchanged) [131], a trusted third party that is needed during every run of the protocol may still result in a performance bottleneck.

In optimistic protocols, there is an offline trusted third party. It is only invoked if something goes wrong. At first, the parties try to perform the exchange on their own, and if both parties follow the protocol (and no message sent between them gets overly delayed), the parties will quickly terminate the protocol. Otherwise, a party that suspects that it is cheated can invoke the trusted third party, which will then either resolve (i.e., make sure that both parties receive their item) or abort (i.e., ensure none does) [6].

Recently, attempts have been made to replace the trusted third party by a trusted computing base inside the parties [8], but these protocols have not yet reached the performance and security standards of a distinguished TTP.

### 4.2 Private Matching

Private matching is a class of problems in secure computation where two or more parties determine whether their private inputs and/or some shared inputs satisfy some specific constraints. Well-known examples of private matching are Yao’s classical millionaires problem [242], the related problem of string matching [103, 121] or the socialist millionaires problem [161, 34] and plaintext-equality testing [165]. More advanced examples, see below, involve operations on privately held databases, say, where two parties like to compute the intersection of their databases, and more generally to enable privacy-preserving datamining. Protocols for private matching serve as building blocks for auction schemes and election schemes, as well as for schemes involving searching and sorting of (encrypted) data.

An interesting application is matching of biometric profiles, e.g., by securely determining whether the (Hamming) distance between a stored biometric template and a sampled one is sufficiently small. However, fully secure versions of such matching protocols are not very efficient yet. A possible way-out is to apply protocols for secure approximation of the Hamming distance between two vectors of length $\ell$, say, which is of sublinear complexity $o(\ell)$, see [122]. (See also subsequent work such as [151]). Recent work, such as [133] includes approximation techniques for private matching as well. Rather than computing the exact result by means of a secure computation, it may be feasible and sufficient to compute an approximate result. The same approach extends to what might be called “approximate datamining in encrypted databases”. The general context for this problem is called privacy-preserving datamining, where the primary task is the development of models (statistics) about aggregated data rather than providing exact matches with individual data records. See, e.g., [182, 1, 133].
More precisely, a private matching protocol PM is two-party protocol between a client \( C \) (called the chooser) and a server \( S \) (called the sender). \( C \)'s input consists of a set \( X = \{x_1, \ldots, x_{k_C}\} \) and \( S \)'s input consists of a set \( Y = \{y_1, \ldots, y_{k_S}\} \). Both sets contain elements drawn from a large domain. At the end of protocol \( C \) learns \( X \cup Y \).

The implementation of \cite{133}. The implementation is based on a semantically-secure public-key encryption scheme \( \text{Enc} \) such that:

1. Given two encryptions \( \text{Enc}(m_1) \) and \( \text{Enc}(m_2) \), it is possible to efficiently compute \( \text{Enc}(m_1 + m_2) \).
2. Given encryption \( \text{Enc}(m_1) \) and a constant \( c \) it is possible to efficiently compute \( \text{Enc}(cm_1) \).

Paillier’s cryptosystem \cite{202} is a candidate for such a homomorphic cryptosystem. The construction of \cite{133} uses the fact that the two properties above imply that it is possible, given encryptions of the coefficients of a polynomial \( P \) and a point \( y \), to compute an encryption of \( P(y) \).

The protocol presented in \cite{133} can be described as follows. \( C \) defines protocol

\[
P(y) = (x_1 - y)(x_2 - y) \cdots (x_{k_C} - y)
\]

and sends an encryptions of the coefficients \( a_0, \ldots, a_{k_C} \) of \( P \) to \( S \). For each \( y_i \in Y \), \( S \) picks a random \( r \) and sends and encryption of \( r \cdot P(y_i) + y_i \). Now \( C \), will receive an encryption \( y_i \) for each \( y_i \in X \cup Y \) and an encryption of a random value for each \( y_i \in Y \) which is not in \( X \).

The communication complexity is \( O(k_C + k_S) \) and the protocol requires \( O(k_C + k_S \log \log k_C) \) by using multiple low-degree polynomial and hashing.

The protocol described above is the building block for related functionalities like: private matching for set cardinalities (in which the chooser only learns the cardinality of the intersection) and private matching for cardinality threshold in which the chooser only learns whether the intersection is larger than a given threshold.

The implementation of \cite{2}. Let \( E \) and \( E' \) be a pair of encryption functions such that \( E \) is known only to \( C \) and \( E' \) is known only to \( S \) and, for all \( x \), \( E(E'(x)) = E'(E(x)) \). For example, \( E \) and \( E' \) could be defined as \( E(x) = x^e \mod p \) and \( E'(x) = x^{e'} \mod p \), where the exponents \( e \) and \( e' \) are known privately to \( C \) and to \( S \), respectively, and \( p \) is known to both.

The protocol is the following.

1. \( C \) computes \( E(X) = \{E(x_1), \ldots, E(x_{k_C})\} \) and sends it to \( S \).
2. \( S \) computes

\[
E'(Y) = \{E'(y_1), \ldots, E'(y_{k_S})\}
\]

and

\[
E'(E(X)) = \{E'(E(x_1)), \ldots, E'(E(x_{k_C}))\}
\]

and sends them to \( C \).
3. \( C \) computes \( E(E'(Y)) \) which is equal to \( E'(E(Y)) \) and, from this list and \( E'(E(X)) \), \( C \) recovers the intersection \( E'(E(X)) \cup E'(E(Y)) = E'(E(X \cup Y)) \) and sends it to \( S \).
4. \( S \) computes \( E(X \cup Y) \) from which \( C \) computes the intersection.
4.2.1 Keyword search

In the context of secure access to large databases, secure keyword search is a fundamental operation.

More specifically, the system consists of a server $S$ and a client $C$. The server’s input is a database $X$ of $n$ pairs $(x_i, p_i)$, each consisting of a keyword and a payload. Key-words can be strings of an arbitrary length and payloads are padded to some fixed length. We may also assume, without loss of generality, that all $x_i$ are distinct. The client’s input is a searchword $w$. If there is a pair in the database in which the keyword is equal to the searchword, then the output is the corresponding payload. Otherwise the output is a special symbol $\bot$. Private keyword search (KS for short) requires privacy for both the client and the server, i.e., neither party learns anything more than is defined by the above transaction.

The following protocol for KS has been presented in [134].

1. The server defines $L$ bins and maps the $n$ items into the $L$ bins using a random, publicly-known hash function $H$ with a range of size $L$. $H$ is applied to the database’s keywords, i.e., $(x_i, p_i)$ is mapped to bin $H(x_i)$. Let $m$ be a bound such that, with high probability, at most $m$ items are mapped to any single bin.

2. For every bin $j$, the server defines two polynomials $P_j$ and $Q_j$ of degree $(m - 1)$. The polynomials are defined such that for every pair $(x_i, p_i)$ mapped to bin $j$, it holds that $P_j(x_i) = 0$ and $Q_j(x_i) = (p_i | 0^\ell)$, where $\ell$ is a statistical security parameter.

3. For each bin $j$, the server picks a new random value $r_j$ and defines the polynomial $Z_j(w) = r_j \cdot P_j(w) + Q_j(w)$.

4. The two parties run an oblivious polynomial evaluation protocol (this can be based on Paillier’s cryptosystem [202] as in the previous section) in which the client evaluates all $L$ polynomials at the searchword $w$.

5. The client learns the result of $Z_{H(w)}(w)$, i.e., of the polynomial associated with the bin $H(w)$. If this value is of the form $p|0^\ell$ the client outputs $p$, otherwise it outputs $\bot$.

This protocol has communication overhead $O(\text{poly log } n)$. In the paper [134], a different construction based on oblivious pseudorandom functions is also presented.

4.3 Anonymous Credentials

In an anonymous credential system it is desired that the service provider cannot link a request for the service with a specific user or with other past requests. We list the main properties that an anonymous credential system should enjoy.

1. Security: it is hard for a coalition of users to get access to a service without having the requested credentials.

2. Multi-show privacy: a user during a transaction can prove possession of credentials and, at the same time, the service provider does not obtain any private user information. This holds even if the user interacts using the same credential certificate several times with the same (or other) service provider.
3. **Usability:** a user that possesses a credential certificate should be able to prove *general statements* (for instance the satisfaction of linear Boolean formulae) over the credentials while preserving multi-show privacy.

4. **Non-Transferability:** it should be inconvenient for a user to lend his credentials to another user.

5. **Efficiency:** the overhead in terms of communication and computation imposed by the anonymous credential system to users and service providers must not heavily affect their performance.

An anonymous credential system [82] can be based on the concept of proofs in which a user shows possession of some piece of information (the credentials) that satisfies some given conditions (e.g., the access control policy). A first implementation of these proofs, for the case in which the conditions are expressed by a monotone Boolean formula, can be traced back to the general results on Sigma protocols by [96] and on statistical zero-knowledge proof systems by [111]. In [41, 40] these techniques are further explored and their applicability to real-life scenarios shown. In particular, Brands [41] presented a Public-Key Infrastructure in which a user can prove in zero knowledge that the credentials encoded by his certificate satisfy a given linear Boolean formula. Brands’ constructions are efficient since only a few modular exponentiations (linear in the number of encoded credentials) have to be performed in order to prove that the credentials encoded in the certificate satisfy a given linear Boolean formula. The main drawback of Brands’ certificates is that they are *one-show* in the sense that using the same certificate in two distinct transactions links the two transactions as performed by the same user. As a consequence, a user needs to obtain from the trusted authority an impractically large batch of certificates so that no certificate is used twice.

In [64] Camenisch and Lysyanskaya proposed an anonymous credential system that is based on the strong RSA assumption and the DDH assumption. In the system of [64, 63] it is possible to unlinkably prove possession of a credential supporting the multi-show property, and the entities that release credentials can independently choose their cryptographic keys. Furthermore, the system allows to encode attributes such as age or expiration into a credential which can be selectively revealed when possession of a credential is proved.

In [239] Verheul proposed a very efficient solution for multi-show credentials, although without proving the security of his system. The result is related to the assumption that for some groups the *Decisional* version of the Diffie-Hellman (DH) problem is easy while its *Computational* version (CDH) is hard and on an additional *ad-hoc* assumption.

Recently, Boneh and Boyen [31] as well as Camenisch and Lysyanskaya [66] have put forth signature schemes that are based on Discrete Logarithm Assumption and, in a similar way that [64] employs a signature scheme [65] based the Strong RSA Assumption, that give rise to a credential system based solely on Discrete Logarithm-related assumption. Finally, Bangerter, Camenisch, and Lysyanskaya [10] describe a generalization of anonymous credential systems and how to construct them from known signature and encryption schemes.

Persiano and Visconti [207] provide a credential system that also allows to encode attributes which is based on a new computational assumption in RSA rings.

**Open Problem No. 15:** Find verifiable encryption schemes that allow for more efficient proofs that an attribute contained in a credential one possesses is encrypted under a third parties public key.
Open Problem No. 16: Construct credential systems (i.e., signature schemes with protocols such as the one by Camenisch and Lysyanskaya [65]) that are secure under alternative or weaker assumptions than the existing ones.

The design of an anonymous credential system is related to the design of group signature schemes. In particular in [66] it is stressed that an anonymous credential system can be designed on top of a set of protocols that is properly included in a set of protocols that achieve group signatures. In recent work of [171], the problem of designing group signature schemes in concurrent settings is discussed and new solutions have been proposed. The impact of these new results for the design of anonymous credential systems that preserve their efficiency and security even when concurrently composed with other protocols is a challenging area.

4.4 Property-Based Attestation

The rapid expansion of world-wide connectivity has changed the requirements on IT systems. We require systems which can guarantee authenticity, integrity, privacy, anonymity, and availability. Cryptography and many other technical security measures such as firewalls, Intrusion Detection Systems and so on are useful tools providing solutions to a variety of security related problems. However, they would work properly only if the underlying computing platform, in particular the operating system, is secure. Existing computing platforms, however, suffer from various security problems due to their architectural weaknesses in hardware and software as well as their complexity [223].

In this context the computing industry has come up with Trusted Computing (TC), a new generation of computing platforms based on new architectures both in hardware and software. The results of their investigations are the two well-known initiatives by the TCG (Trusted Computing Group)\(^1\), an alliance of leading IT enterprises, and Microsoft’s NGSCB (Next-Generation Secure Computing Base)\(^2\). Whereas there is no technical specification for NGSCB available yet, TCG has published the corresponding hardware specifications [237, 236].\(^3\)

Loosely speaking, the basic idea is to embed a “trusted third party” into the underlying hardware where this party is realized by tamper-resistant hardware components. The intention is, however, to keep the tamper-resistance assumption as weak as possible and reduce the cost by keeping the trusted component as small as possible. The stated goal of these architectures is to improve the security and trustworthiness of computing platforms [189, 192, 223, 222]. Indeed, these platforms offer many useful functions which can be used to increase a platform’s security. They extend the conventional PC architecture by new mechanisms to (i) protect cryptographic keys, (ii) generate random numbers in hardware, (iii) authenticate (the configuration of) a platform (e.g., of the BIOS and the TCB), called *attestation*\(^4\), and (iv) cryptographically bind the data to be encrypted to certain information, e.g., to the system configuration and the identifier of the invoking application, called *sealing*.

---

\(^1\)www.trustedcomputinggroup.org
\(^2\)www.microsoft.com/ngscb/
\(^3\)Note that Microsoft’s NGSCB [189, 192] is one concrete instantiation of the TCG specification based on additional hardware extensions such as LaGrande (see http://www.intel.com/technology/security/index.htm). Since the NGSCB/LaGrande approach is based on the functionality provided by the TCG specification, we focus on TCG in the following, although our research work and the solutions achieved are applicable to NGSCB, or any other approach that offers the same functionalities.
\(^4\)Data represented as a hash value, using a cryptographically secure hash function such as SHA-1
For better understanding we take a look at an example: on platform startup a hardware component, called the Core Root of Trust Module (CRTM), hashes the current BIOS and the master boot record (MBR) of the boot device and writes the result into protected registers called Platform Configuration Registers (PCR) which are located in a (tamper-resistant) hardware module called Trusted Platform Module (TPM). The software that is executed in the MBR (e.g., the boot loader) can create a chain of trust by writing a hash value of the software it loads (e.g., the operating system) into another PCR register. We hence call the chain of hash values stored in the PCR as the platform configuration.\(^5\) The TPM can cryptographically sign the current platform configuration using a protected signature key, to attest it to a remote challenger. To ensure that data sent to a platform can only be accessed under a specific platform and platform configuration, a sealing mechanism is provided: a remote instance can encrypt critical data including a demanded configuration under an encryption key whose decryption key is only known to the TPM. The TPM can decrypt the cipher but releases the data only if the current platform configuration matches the demanded one. Note that sealing functionality can be used to ensure that the critical data cannot be accessed by the user when the configuration of the platform changes after the attestation took place.

However, there is still an ongoing public debate about the negative economical, social, and technical consequences of these platforms \[4, 5, 227\]. People are concerned about the potential dangers that can be caused by the capabilities of such platforms: they may give vendors and content providers too much control over personal systems and users’ private information. Although most complaints about trusted computing are speculative\(^6\), it is highly important to observe its development carefully, and to improve this technology by putting together the missing pieces for more secure platforms in future. The public debates of the past highlighted several deficiencies and unsatisfying properties of the current TCG specification version 1.1b \[237\]. In October 2003, the TCG published an updated TCG specification version 1.2 \[236\] which solves some of these deficiencies. For instance, a cryptographic protocol called Direct Anonymous Attestation (DAA) \[62\] was specified that, roughly spoken, provides users with an unlimited number of pseudonyms without requiring a trusted Privacy Certification Authority (privacy-CA) that was required (and criticized) in version 1.1b of the specification.

Currently ongoing research work considers deficiencies of TCG regarding the two important functionalities attestation and sealing. The main motivation is that existing proposals for realization of these functionalities allow a remote instance to discriminate certain platforms having certain configurations (of the hardware and software running on the platform). Thus, a remote instance, attesting a platform, is able to exclude certain configurations from his/her business model, e.g., configurations related to alternative operating systems such as Linux. Hence, if in the future attestation is used to enforce the software configuration, powerful vendors may enforce their policies on their products by preventing alternative software products from running on their platforms.\(^7\) Further, the existing proposals enable an instance

\(^{5}\)Note that here we do not mean the hash value of the history but rather the hash of the TCB (trusted computing base) state that remains unchanged during the run-time in contrast to, e.g., history measurements done in \[224\].

\(^{6}\)As pointed out by \[220\] the TC functionalities are completely under the control of the underlying operating system, and as a consequence, users can benefit from the security features of TCG/NGSCB, as long as their operating system is trustworthy.

\(^{7}\)Note that similar approaches can be observed today, e.g., many banks provide banking software for only one operating system, a lot of websites support only one web-browser, and a lot of hardware devices, like music players, expect a specific operating system. However, in conjunction with TC technology, the use of
attesting a platform or observing the attestation to obtain complete information about the hardware and software configuration\(^8\) of a platform making attacks on a platform easier.

Other problems related to attestation are updates and backup: the new functionalities allow to seal critical data (e.g., documents, content) to a certain platform configuration. This, however, strongly limits the usage flexibility when system updates (e.g., patches) change the system configuration. As a consequence, the data is not accessible anymore. Similar situations arise when a system backup is made. It is not possible to access the data on another platform having a different configuration (even if this platform satisfies the same requirements as the previous one). Thus, also the sealing function can be applied to limit or prevent the use of certain products.

Clearly, the above mentioned problems lead to an unsatisfactory situation which is not what we understand under security for all involved parties (in the sense of multilateral security) and under an open market.

In [221] a new approach has been taken which uses these functionalities only based on the properties a platform offers and not based on the configuration of its software and hardware components.\(^9\) A property of a platform describes an aspect of the behavior of that platform regarding certain requirements, such as security-related requirements. In general, platform properties of different abstraction levels are imaginable: a platform property may, e.g., state that a platform has built-in measures conform to the privacy laws, or that it strictly separates processes from each other, or that a platform has built-in functionalities to provide Multi-Level Security (MLS) and so on. The question of whether there is a correct or useful property set depends heavily on the underlying application and its requirements on the environment. For instance a useful property is what is now accepted as secure operating system providing isolation of processes or confinement etc. Hence, different platforms with different components may have different configurations while they may all offer the same properties and consequently fulfill the same requirements.

As a starting point several solutions have been sketched in [221] to the mentioned problems following the property-based attestation paradigm: These solutions (i) differ in their trust models, efficiency and the functionality offered by the trusted components, (ii) can be applied to all approaches that provide some kind of secure booting or application authentication. Further, it is also demonstrated, how the Trusted Software Stack (TSS), the TPM-library proposed by the TCG, can be extended by a property-based attestation protocol based on the existing TC hardware, called Trusted Platform Module (TPM), without a need to change the underlying trust model.\(^10\)

Note that the anonymity provided by DAA or privacy-CA’s is completely orthogonal to the stated goals of property-based attestation. Nevertheless, both approaches can be combined into an anonymous property-based attestation function [221].

---

\(^8\)One may think that the subject of attestation is only to determine a “known” configuration. However, if this would be the case then only a binary confirmation would suffice indicating that a known configuration has been changed. This solution would be much more simpler than the attestation procedure specified by the TCG.

\(^9\)A similar approach has been proposed independently in [211].

\(^10\)It should be noted that our primary goal is to have a non-discriminating attestation as a standard, which can be certified by trusted entities, and on which the vendors and developers of related products should rely. Clearly, standards leave some space for corresponding implementations, and this may open the door for information flow allowing, e.g., operating system footprinting (see, e.g., www.insecure.org/nmap). However, this is not the subject of this paper.
4.5 Group Key Exchange

The establishment of group keys is fundamental for a variety of security mechanisms in group applications. Two different classes of protocols can be identified: (group) key transport (GKT), in which the key is chosen by a single party and transmitted to the other parties via secure channels, and (group) key exchange (GKE), in which all parties interact and all parties contribute towards the computation of the key. In GKE protocols, no secure channels are needed and, more important, no party is allowed to choose the key on behalf of the group: in other words, group members do not trust each other.

In case of group key establishment protocols it is common to distinguish between passive adversaries eavesdropping the communication channel and active adversaries modifying and injecting messages. Additionally, security of group key establishment protocols may depend on whether malicious participants who may deviate from the protocol specification are tolerated or not. In the following we distinguish between security requirements that are common for GKT and GKE protocols and those that are specific to GKE protocols only.

The requirements addressing key secrecy [233, 114, 53] can be seen as fundamental for any group key establishment process. They state that an adversary which is considered to be a non-legitimate protocol participant should not be able to distinguish the group key computed by legitimate participants from a fake, randomly chosen value. These requirements should also hold against active adversaries trying to perform various impersonation attacks in spirit of entity authentication [19] and (implicit) key authentication [188], or getting access to real group keys computed in different sessions in spirit of known-key security [241, 52]. Additionally, secrecy of keys computed in past sessions should hold even in case where the adversary obtains long-term key material of legitimate users at some later stage where it can start to impersonate these users. This is known as (perfect) forward secrecy [150, 114]. All of the just mentioned requirements have been formalized in a single definition of authenticated key exchange (AKE) security, originally in the formal security model in [47] and with subsequent variants and refinements in [46, 170, 169, 49]. A general technique, a compiler, to achieve AKE-security from any “passively secure” GKE protocol has been suggested in [170] and later analyzed and improved in [49].

The second major block of security requirements for both GKT and GKE protocols address the correctness of the group key establishment protocol. The notions of key confirmation [188] and mutual authentication [19] result in the inability of the adversary to influence legitimate protocol participants computing different group keys while believing that all other legitimate participants were involved in the actual protocol execution. As noted in [35] these requirements are helpful to prevent unknown key-share attacks [114] by which an adversary tries to make a protocol participant believe that the key is shared with some specific (legitimate) party when it is in fact shared with another party (not necessarily the adversary itself). From the perspective of formal security models for group key establishment the informally described requirements of key confirmation an mutual authentication have been formalized as mutual authentication (MA) security in [47] for the case where all parties are assumed to be honest and extended in [169, 49] for the case of malicious participants following the observations of [86], and in [48] for the case of strong corruptions following the ideas of [229] for two-party key exchange. Also [169, 49] describe general compilers for achieving MA-security from any AKE-secure group key exchange protocol.

The last major block of security requirements concerns the question of key control [190], i.e., attacks by which an adversary either an outsider or a legitimate protocol participants
influences the final value of the group key. Related notions called *contributiveness* and *unpredictability of group keys* [7] require that all protocol participants equally contribute to the computation of the group key in order to guarantee its freshness. A similar notion named *security against key-replication attacks* [177] prevents the occurrence of the same group key in two different sessions. As noted in [48] these requirements implicitly state the difference between GKT and GKE protocols – namely, that in GKE protocols none of the participants is trusted to choose the group key on behalf of other participants, whereas the party responsible for the choice of the group key in GKT protocols has obviously power to choose any value of its choice. Several aspects of key control issues have also been informally addressed in [45] and [174], then formalized in [30] and strengthened in [48] by considering strong corruptions, however, without focusing on the ability of the adversary to influence the probability distribution of the computed group key. I.e., the approach in [48] prevents the adversary from enforcing some particular value as a group key, but not from biasing a subset of its bits. Additionally, [48] described a general compiler for contributiveness of GKE protocols in case of strong corruption attacks.

**Open Problem No. 17:** Achieve indistinguishability of group keys computed in the presence of malicious participants under the condition that strong corruption are possible. In order to solve this problem, it might be necessary to assume certain bounds on the number of malicious parties.

A more detailed analysis of security models for group key exchange protocols w.r.t. the mentioned blocks of security requirements is available in [186]. Another survey focusing on concrete group key exchange protocols and analyzing their security requirements can be found in [185].

In addition to the consideration of the above mentioned security requirements, the aspects of *fault-tolerance* and *robustness* due to the distributed nature of group key exchange protocols are also of prime importance. It might be of great benefit if protocol participants can compute the group key even if some of them crash, either deliberately or due to the occurring network connection failures. One of the first solutions for fault-tolerant GKE protocols providing forward secrecy, has been proposed in [61] using techniques of an *asynchronous distributed consensus protocol*, e.g., [79, 59]. A follow-up proposal in [113] addressed robustness under consideration of malicious protocol participants and, in particular, focusing on authentication and key control issues, however, without focusing on strong corruptions.

**Open Problem No. 18:** Design a group key exchange protocol which provides fault-tolerance and robustness as well as the requirements of AKE-, MA-security, and prevents key control, while considering malicious participants and strong corruptions.

**Open Problem No. 19:** Design a generic compiler for fault-tolerance and robustness of independently designed secure group key exchange protocols.

### 4.6 Multiparty Computation and Economy

Doing business often involves confidential information as well as parties that have potentially conflicting interests and no prior reason to trust each other. Multiparty computation protocols can be very helpful in such a scenario.
Economic models are also the basis for the broader field of rational protocols in cryptography. They are the subject of a related ECrypt report [199].

4.6.1 Auctions

An example scenario motivated by economic theory is a double auction. Here, some commodity is being traded on a market with many potential sellers and buyers. Each party has some intentions initially. A buyer, for instance, will for each quantity have a maximum price in mind that he is willing to pay for this quantity. Such information is of course confidential: if buyers would make this information public, it would be an incentive for the sellers to adjust their prices to their own advantage. Similarly, sellers need to hide at which price they are willing to deliver a certain quantity of a goods.

In economic theory, one can show a theorem that holds for a very general class of trading mechanisms, including double auctions. In a nutshell, the theorem says that if the confidential information of all players was given to a trusted third party, it could compute a solution to the given scenario that is always at least as good as what could be obtained by direct negotiation between players. In the case of double auctions, the solution is the so called market clearing price, the price per unit of the commodity that will enable trading the maximum value, and hence give the maximum joint benefit for the players.

Of course, establishing such a third party which players can trust is in most cases either impossible or very expensive. However, the definition of secure multiparty computation exactly says that a secure protocol for computing, e.g., the market clearing price is equivalent to having a trusted third party as required.

Open Problem No. 20: Design special-purpose secure multiparty protocols for handling economic scenarios such as auctions. Develop this promising and very interesting research field, which may lead to development of new techniques and protocols that are useful in practice.

Progress towards a solution to the auction problem has been made by Bogetoft et al. [29]. They consider a double-auction implementation for trading production rights to a processor of farm products. As an important buliding block for such auctions, a new constant-round support protocol for oblivious computation with shared values, has recently been found [106].

4.6.2 Fair Division

Consider a situation where \( n \) players \( A, B, C, \ldots \) have to divide a set of goods \( G_1, G_2, \ldots, G_k \) among themselves. Every player \( G_i \) has its own, private valuation \( V_{i,1}, V_{i,2}, \ldots, V_{i,k} \) of all goods. The goods may be divisible and may be allocatable in parts to the players (such as cutting a cake at a coffee table) or may be indivisible and have to be assigned as a whole (such as a house and a car in the case of a divorce). When divisible, the goods may be homogeneous and can easily be divided, respecting the valuations, or they may be heterogeneous, which makes it harder to split them.

The goal is to split the goods in a fair way such that every player receives a share of the goods that she considers to be a “fair” share of the sum. This is not trivial because every player has its individual valuation. There are several ways to define fairness:

**Proportionality:** Every player receives a share that it values at least \( 1/n \) of the total value.
Envy-freeness: Every player values its own share at least as much as the share of any other player.

Efficiency (Pareto-optimality): There is no other division that is strictly better for one player and at least as good for all others.

Equitability: The (announced) valuations of the shares of all parties are equal.

There exists a rich literature on the subject with practical applications to conflict resolution in politics as well as in court cases (see [37] and references therein). A particular procedure for dividing a set of divisible homogeneous goods among two players is the Adjusted Winner method of Brams and Taylor [39]. It provides envy-freeness, efficiency, and equitable divisions and therefore achieves an optimal degree of fairness for two parties, given the above notions of fairness. It can be implemented by a trusted third party, who receives the valuations of every player, computes a ranking of the goods according to the relative valuations of the two players, and continues to assign shares to the players based on the ranking.

Another recent method is the Surplus Procedure [38] that works for splitting one continuous heterogeneous item, which is typically modeled as a cake (the source of the name “cake cutting” for this area of mathematical problems). The Surplus Procedure is strategy-proof, an interesting property which means that no player can get an advantage by not announcing a truthful valuation as input to the protocol. To compute the division with the Surplus Procedure, a trusted first assigns to each player a share that the player values at 50% of the whole, and then splits the remaining part so that each player receives the same proportion under its respective valuation.

Many division procedures have to be computed by a trusted third party in practice because a player knowing his opponent’s valuation can cheat the method by not using his true valuation for the procedure. This holds in particular for Adjusted Winner and for the Surplus Procedure. Therefore, it is interesting to provide cryptographically secure implementations of fair division methods.

Open Problem No. 21: Design an efficient two-party protocol for securely computing fair divisions, in particular, using the Adjusted Winner method and the Surplus Procedure.

4.7 Electronic Voting

Since the introduction of public-key cryptography at the end of the seventies many cryptographic protocols for electronic voting have been proposed. (Here, electronic voting means that the entire process is electronic, including distribution of the ballots and voter authentication.)

Various classes of solutions have emerged. One interesting class is formed by protocols that assume an anonymous channel is available for the voters (see, e.g., [135] which uses blind signatures). The assumption of an anonymous channel is however quite strong. The resulting protocols are computationally efficient, but fail to achieve the property of universal verifiability of the election result.

The class of solutions based on either threshold homomorphic encryption or on verifiable mixes, however, meets the requirement of universal verifiability. Also, no anonymous channels
are needed during voting (or, any other stage of the protocols). The notion of universal verifiability first appears in the work by Benaloh et al. [90, 25], followed by much more efficient solutions in [97] and [98], where the latter paper actually introduces the use of a threshold homomorphic encryption (rather than using verifiable secret sharing). Work on verifiable mixes started in [225], leading to the first linear-time verifiable mixes as achieved in [197] and in [136], and further improvements such as [149].

Another class of solutions adds voter verifiability property to universal verifiability, leading to end-to-end verifiable protocols. This properties encompasses the idea that a voter can verify that her vote was cast as intended and tallied as cast. This is usually achieved through paper receipt, given to the voter when her vote is cast. After the casting period, all receipts are shuffled and published on a public bulletin board. The final tally is computed from this bulletin board. This allows any voter to verify her vote was tallied properly (see, e.g., [85, 210, 126]).

An interesting issue is the problem of incoercible elections, which amounts to protecting the voters against coercion. Work on this problem started with [24], who introduced the notion of a receipt-free voting scheme and proposed a scheme using a two-way untappable (i.e., private) channel. An improved proposal appeared in [226], using only a one-way untappable channel (directed from the voting server to the voter). An extension to the more general setting of incoercible multiparty computation was introduced in [72], which proposes to use deniable encryption (see [70]) as tool to achieve incoercibility.

Quite a few papers appeared since. The best solutions known to date for achieving some level of incoercibility employ so-called randomizers, after [158] and improved in [13]. The known solutions, however, are not satisfactory though. Firstly, the assumption of an untappable channel cannot be achieved for common networks such as the Internet. Secondly, solutions based on randomizers require a voter to know which ones have been corrupted. More generally, an exact definition and model for receipt-freeness and incoercibility has not fully been established yet. Some work in this direction can be found in [191, 164, 112, 163].

**Open Problem No. 22:** Provide an exact, formal definition of receipt-freeness and incoercibility, either in terms of standard cryptographic definitions, or, preferably, in terms of one of the frameworks for universally composable cryptographic protocols.

**Open Problem No. 23:** Either eliminate the assumption of an untappable channel from electronic voting protocols, or prove that untappable channels are necessary.
Bibliography


of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, pages

[15] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic ad-
versaries. In R. A. Rueppel, editor, Advances in Cryptology - EUROCRYPT ’92, pages
658.


[17] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and
analysis of authentication and key exchange protocols. In Proceedings of the Thirtieth
Annual ACM Symposium on the Theory of Computing, pages 419–428, Dallas, TX,

Rity for public-key encryption schemes. In H. Krawczyk, editor, Advances in Cryptology
Science Volume 1462.

in Cryptology–CRYPTO’93, volume 773 of Lecture Notes in Computer Science, pages

abstract). In Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory


[22] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-
cryptographic fault-tolerant distributed computation (extended abstract). In Proceed-
ing of the Twentieth Annual ACM Symposium on Theory of Computing, pages 1–10,

[23] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computation with optimal


[25] J. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy
of voters. In Proc. 5th ACM Symposium on Principles of Distributed Computing (PODC
’86), pages 52–62, New York, 1986. A.C.M.


