Player Simulation and General Adversary Structures in Perfect Multiparty Computation*

Martin Hirt and Ueli Maurer
Department of Computer Science, ETH Zurich,
CH-8092 Zurich, Switzerland
{hirt,maurer}@inf.ethz.ch

Communicated by Oded Goldreich

Received 31 December 1997 and revised 26 February 1999

Invited to the Journal of Cryptology, 13(1):31–60, 2000.

Abstract

The goal of secure multiparty computation is to transform a given protocol involving a trusted party into a protocol without need for the trusted party, by *simulating* the party among the players. Indeed, by the same means, one can simulate an arbitrary player in any given protocol. We formally define what it means to simulate a player by a multiparty protocol among a set of (new) players, and we derive the resilience of the new protocol as a function of the resiliences of the original protocol and the protocol used for the simulation.

In contrast to all previous protocols that specify the tolerable adversaries by the number of corruptible players (a threshold), we consider general adversaries characterized by an adversary structure, a set of subsets of the player set, where the adversary may corrupt the players of one set in the structure. Recursively applying the simulation technique to standard threshold multiparty protocols results in protocols secure against general adversaries.

The classical results in unconditional multiparty computation among a set of n players state that, in the passive model, any adversary that corrupts less than $n/2$ players can be tolerated, and in the active model, any adversary that corrupts less than $n/3$ players can be tolerated. Strictly generalizing these results we prove that in the passive model, every function (more generally, every cooperation specified by involving a trusted party) can be computed securely with respect to a given adversary structure if and only if no *two* sets in the adversary structure cover the full set of players, and in the active model, if and only if no *three* sets cover the full set of players. The complexities of the protocols are polynomial in the number of maximal adverse player sets in the adversary structure.

Key words. Multiparty computation, Information-theoretic security, Player simulation, General adversaries, Adversary structures.

*Research supported by the Swiss National Science Foundation (SNF), SPP project no. 5003-045293. Some of the results in this paper have been published in [HM97].

1 Introduction

1.1 Secure Multiparty Computation

Consider a set of players who wish to cooperate in a specified manner but do not trust each other. Assume that the cooperation could be realized if a mutually trusted party was available, but that no such trusted party exists. In other words, the specification of the desired cooperation can be given in terms of a protocol among the players and a trusted party, and the goal of the multiparty computation between the players is to perform the same cooperation without the trusted party, where security is guaranteed if the subset of players that cheat is not too large. This is achieved by a protocol that simulates the trusted party.

One particular type of cooperation is the computation of an agreed function of the players' inputs in a secure way (secure function evaluation). Most papers in the literature are restricted to this scenario, but their results often apply also to the general model discussed above. Secure function evaluation is trivial when a trusted party τ is available: every player sends his input to τ who computes and announces the result.

An example of a general secure cooperation scenario is to simulate a fair stock market among a set of participants (investors) without need for a trusted stock exchange system. The major difference between secure function evaluation and general secure cooperation is that in the latter, inputs may be given not only at the beginning but also during the entire computation.[1]

Security of a multiparty protocol is defined with respect to a central adversary that may corrupt certain players (corresponding to the collaborating cheating players). For secure function evaluation, security means that the output of the computation is guaranteed to be correct (correctness) and that the players' inputs remain private (privacy), even when the corrupted players misbehave. For a general cooperation protocol, security means that whatever the adversary can achieve in the cooperation protocol can also be achieved directly in the specification involving the trusted party (without corrupting the trusted party).

1.2 Classification

One generally distinguishes between *passive* and *active* adversaries. A passive adversary can read all information available to the corrupted players and tries to violate the privacy, but not the correctness, of the computation. In contrast, an active adversary controls the complete behavior of the corrupted players, trying to violate the privacy and/or the correctness of the computation.

The communication models differ with respect to three criteria: whether or not pairwise *secure communication channels* are available, whether or not *broadcast* channels are available, and whether the communication channels are *synchronous* or asynchronous.[2] The models with secure communication channels are also referred to as *secure channels models*.

Adversaries can be classified according to whether their computational resources are polynomially bounded (*cryptographic* security), or unbounded (*unconditional* or information-theoretic security).

[1] A general secure cooperation could also be reduced to secure function evaluation by having each player give his strategy as input to the function. However, this approach would require that every player is totally aware of his strategy, and also that the environment of the computation can be modeled. As an example, each time a player is required to give some input to a general secure cooperation, he could enter the number of hits when searching some specific term with an internet search engine. Modeling this strategy would require to model (at least) the behavior of all internet users.

[2] Synchronous means that the delay of messages is bounded by a constant. See [Can95] for more details.

Clearly, unconditional security can only be achieved in a secure channels model. In the unconditional model one can distinguish between protocols with exponentially small or with zero failure probability. We refer to the latter as *perfect* multiparty computation. Further, one distinguishes between *static* and *adaptive* (or dynamic) adversaries. In contrast to a static adversary that corrupts players at the beginning of the protocol execution, an adaptive adversary is allowed to enlarge the set of corrupted players during the protocol execution, as long as the total set of corrupted players remains admissible. Recently, *mobile* adversaries were also considered (e.g., [OY91] and [CH94]). As an adaptive adversary, a mobile adversary can corrupt players at any time, but he can also "release" corrupted players, regaining the capability to corrupt further players. Security against a mobile adversary is referred to as "pro-active security."

1.3 Previous Work

The problem of general-purpose multiparty computation was first stated by Yao [Yao82]. As a first general solution to this problem, Goldreich, Micali, and Wigderson [GMW87] presented a passively secure protocol that allows n players to securely compute any given function even if a passive adversary corrupts any $t < n$ players, and an actively secure protocol that tolerates an active adversary corrupting any $t < n/2$ of the players. The security of the protocols is cryptographic, that is the adversary is assumed to be polynomially bound. Chaum, Damgård, and van de Graaf [CDG87] improved the bound for the active model in the sense that the input of one player can even be information-theoretically hidden. Galil, Haber, and Yung [GHY87] considered efficiency and several corruption types in the cryptographic model. Ben-Or, Goldwasser and Wigderson [BGW88] proved that in the secure channels model without broadcast, perfect security for n players can be achieved even if the adversary can corrupt any set of less than $n/2$ players (passive case) or, alternatively, any set of less than $n/3$ players (active case). These bounds are tight. The same results were obtained independently by Chaum, Crépeau and Damgård [CCD88] in an unconditional model with exponentially small error probability. The bound for the active model was improved by Rabin and Ben-Or [RB89] by assuming a broadcast channel and tolerating a negligible error probability. They proposed protocols that provide unconditional security against an active adversary that may corrupt any $t < n/2$ of the players. This result was also achieved by Beaver [Bea91b] with higher efficiency. Combining the advantages of unconditional security (against an adversary that corrupts a certain fraction of the players) and cryptographic security (against an adversary with limited computing power), Chaum [Cha89] presented a protocol which tolerates an active adversary that either corrupts at most $t < n/3$ players, or is polynomially bounded.

The types of tolerable adversaries have recently been generalized in a number of directions (adaptive adversaries, e.g., [CFGN96], uncoercibility, e.g., [CG96], combined active, passive, and fail-adversaries, e.g., [FHM98]), and some authors have investigated multiparty computation for various minimality and complexity criteria, e.g., [Kus89], [BB89], [Bea89], [FY92], [FKN94], [Rab94], [CGT95], and [CKOR97].

Another line of research is concerned with protocols that are tailored to a particular function like voting (e.g., [CFSY96]), auctioning (e.g., [FR96]), sharing of encryption or signature operations (e.g., [dDFY94], [GJKR96a], and [GJKR96b]), or private information retrieval (e.g., [CGKS95] and [KO97]). The major reason for designing protocols for special functions compared with applying a general-purpose protocol is the potential gain of efficiency.

1.4 Contributions of this Paper

This paper is concerned with protocol generators, which for any given function (more generally, for any given specification) generate a protocol for securely computing it. The provided security is perfect (with zero error probability), i.e., we consider a passive or an active adversary with unbounded computing power, in the classical model with pairwise synchronous secure communication channels between players, but not assuming a broadcast channel (like in [BGW88] and [CCD88]). Although the proofs only consider static adversaries, the protocols actually provide security against adaptive adversaries as well. Formal proofs for adaptive security are beyond the scope of this paper.

All previous results in the literature specify the sets of potentially corrupted players by their cardinality, i.e., by a threshold. We define more generally the security of a multiparty computation protocol with respect to an *adversary structure*, a monotone set of subsets of the players, where the adversary may corrupt the players of *one* set in this adversary structure. An adversary structure is monotone in the sense of being closed with respect to taking subsets, and corresponds to the notion of an access structure in the area of secret sharing (or, more precisely, the complement of it), e.g., [ISN87] and [BL88]. Note that which particular subset is corrupted is not known in advance, and in fact may even remain unknown after the protocol execution.

As an example of an adversary structure, consider the set P of players and the adversary structure $\mathcal{Z}$, where

$$\begin{aligned} P &= \{p_1, p_2, p_3, p_4\}, \\ \mathcal{Z} &= \Big\{\emptyset, \{p_1\}, \{p_2\}, \{p_3\}, \{p_4\}, \{p_1, p_2\}, \{p_1, p_3\}, \{p_1, p_4\}\Big\}. \end{aligned}$$

In this example, the adversary can choose either to corrupt no player, or to corrupt a single player, or to corrupt p_1 and an additional player.

The contributions of this paper are twofold: First, we propose a framework for constructing new secure multiparty computation protocols by simulating players in known protocols. In particular, we make explicit the concepts of a specification and a protocol generator which converts a specification into a protocol, in contrast to [GMW87] and [BGW88] where tools for constructing protocols are described and these new concepts are only implicit. Simulating a player by a set of players means to perform all operations of the simulated player by a multiparty protocol among the simulating players. Of course, any of the simulating players can again be simulated. The adversary structure tolerated by the resulting protocol is derived and proven based on the tolerated adversary structures of the basic protocol and the simulation protocol.

Second, we introduce the notion of general adversary structures into the field of multiparty computation, and we characterize exactly which adversary structures can be tolerated in information-theoretically secure multiparty computation. For a given set P of players and an adversary structure $\mathcal{Z}$, we define $Q^{(2)}(P, \mathcal{Z})$ to be the predicate that no two sets in $\mathcal{Z}$ cover the full set P of players, and we define $Q^{(3)}(P, \mathcal{Z})$ to be the predicate that no three sets in $\mathcal{Z}$ cover the full set P of players. Formally

$$\begin{aligned} Q^{(2)}(P, \mathcal{Z}) &\iff \forall Z_1, Z_2 \in \mathcal{Z} : Z_1 \cup Z_2 \neq P, \\ Q^{(3)}(P, \mathcal{Z}) &\iff \forall Z_1, Z_2, Z_3 \in \mathcal{Z} : Z_1 \cup Z_2 \cup Z_3 \neq P. \end{aligned}$$

The following tight bounds are proved:

1. In the passive model, as a strict generalization of the threshold-type result of [BGW88] and [CCD88], perfect multiparty computation for any function (specification) is possible if and only if no two sets in the adversary structure cover the full player set (i.e., $Q^{(2)}(P, \mathcal{Z})$ is satisfied).

2. In the active model, as a strict generalization of the threshold-type result of [BGW88] and [CCD88], perfect multiparty computation for any function (specification) is possible if and only if no three sets in the adversary structure cover the full player set (i.e., $Q^{(3)}(P, \mathcal{Z})$ is satisfied).

In general, the threshold-type structures are not maximal and hence our protocols can tolerate strictly larger adversary structures. For example, in the active model with six players, the protocol of [BGW88] tolerates only adversaries that corrupt at most one player (formally, it tolerates the adversary structure $\mathcal{Z} = \big\{\emptyset, \{p_1\}, \{p_2\}, \{p_3\}, \{p_4\}, \{p_5\}, \{p_6\}\big\}$). Our approach yields protocols that tolerate additional pairs and even triples of potentially corrupted players. For example, the adversary structure

$$\begin{aligned}\mathcal{Z} = \Big\{&\emptyset, \{p_1\}, \{p_2\}, \{p_3\}, \{p_4\}, \{p_5\}, \{p_6\},\\ &\{p_2, p_4\}, \{p_2, p_5\}, \{p_2, p_6\}, \{p_3, p_5\}, \{p_3, p_6\}, \{p_4, p_5\}, \{p_4, p_6\}, \{p_5, p_6\},\\ &\{p_2, p_5, p_6\}, \{p_4, p_5, p_6\}\Big\}\end{aligned}$$

satisfies $Q^{(3)}$ and can hence be tolerated. It is clear that every adversary structure is monotone, and hence it is sufficient to enumerate only the maximal sets. The set of maximal sets of an adversary structure $\mathcal{Z}$ is called the *basis* $\overline{\mathcal{Z}}$. In the above example,

$$\overline{\mathcal{Z}} = \big\{\{p_1\}, \{p_2, p_4\}, \{p_3, p_5\}, \{p_3, p_6\}, \{p_2, p_5, p_6\}, \{p_4, p_5, p_6\}\big\}.$$

When our results are applied to reliable broadcast (Byzantine agreement), they provide the first nonthreshold broadcast protocol, as required for example in [CDM98] (where later solutions [FM98] are more efficient). Applying the results to verifiable secret-sharing, they provide a nonthreshold verifiable secret-sharing scheme as first proposed in [Gen96].

The primary emphasis of this paper is on the existence of protocols. Indeed, all proposed protocols have time and communication complexities polynomial in the number of maximal sets in the adversary structure,[3] but further efficiency considerations and tuning are suggested as future work (however, efficiency polynomial in the number of players is impossible to achieve for all adversary structures, see Theorem 4). Also, concurrency is not addressed in this paper, and our formalism does not support the analysis and/or the optimization of the round complexity of the protocols. The constructions in this paper are based on *oblivious* protocols [BGW88], i.e., on protocols in which the sequence of executed statements is independent of the contents of the previous messages (as opposed to protocols like, e.g., [FM88] and [BB89], in which the flow depends on the contents of the variables), and the proposed formalism is restricted to oblivious protocols.

1.5 Motivating Examples

As a first example, consider a set of five players $P = \{p_1, p_2, p_3, p_4, p_5\}$ that participate in a nine-party protocol of [BGW88] (passive case), where p_3 and p_4 each play for two players, p_5 plays for three players, and p_1 and p_2 each play for one player of this protocol. This protocol tolerates the adversary

[3] The constructions of polynomial protocols are based on joint work with Matthias Fitzi [Fit96].

structure that contains $\{p_1, p_2, p_3\}$, $\{p_1, p_2, p_4\}$, $\{p_1, p_5\}$, $\{p_2, p_4\}$, $\{p_3, p_4\}$ (and of course all subsets of these sets), because every set in this structure plays for at most four players in the 9-party protocol.

More generally, consider a protocol for the set P of players in which each player p_i acts for w_i players in a threshold-type protocol of [BGW88] with $n = \sum_{i:p_i \in P} w_i$ players. In the passive model, security is guaranteed with respect to the adversary structure

$$\mathcal{Z} = \left\{ Z \subseteq P \colon \sum_{i:p_i \in Z} w_i < n/2 \right\},$$

and in the active model, security is guaranteed with respect to

$$\mathcal{Z} = \left\{ Z \subseteq P \colon \sum_{i:p_i \in Z} w_i < n/3 \right\}.$$

These generalized threshold-type protocols are not sufficient for capturing general scenarios of mutual trust and distrust, where players (e.g., people, companies, countries) are often either in a trust relationship (related, married, mutually affiliated, allied) or in a distrust relationship (animosities, competition, hostilities).

As an example of player simulation, consider the set $P = \{p_1, \ldots, p_6\}$ of players, and the four-party protocol of [BGW88] (for the active case) in which p_1 and p_3 play for one player each and the other two players are simulated by four-party protocols of the same type, one among the players p_1, p_2, p_3, and p_4, and the other among the players p_1, p_2, p_5, and p_6 (see Fig. 1).

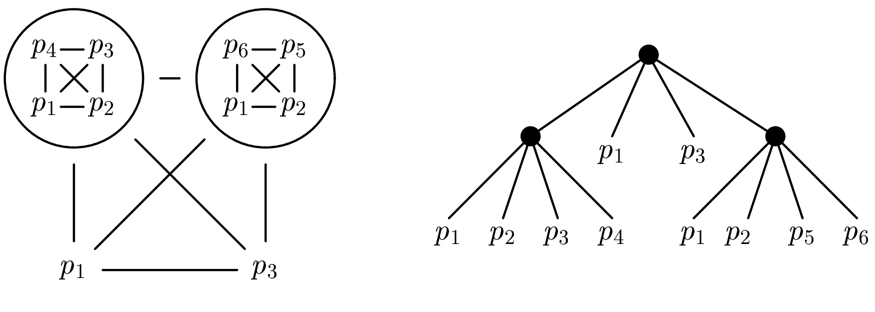

Figure 1: Example of a player simulation.

This protocol tolerates exactly the adversary structure that was discussed in the example in Section 1.4, namely the adversary structure $\mathcal{Z}$ with the basis

$$\overline{\mathcal{Z}} = \big\{\{p_1\}, \{p_2, p_4\}, \{p_3, p_5\}, \{p_3, p_6\}, \{p_2, p_5, p_6\}, \{p_4, p_5, p_6\}\big\}.$$

For each set in $\mathcal{Z}$, one can easily verify that the set is tolerated: for example, the set $\{p_2, p_5, p_6\}$ is tolerated because only one player is corrupted in the simulating protocol among p_1, p_2, p_3, and p_4 (thus this protocol simulates an honest player for the main protocol), and hence three of the four players in the main protocol play honestly. The fact that there are too many corrupted players in the subprotocol among p_1, p_2, p_5, and p_6 does not matter.

The tolerated sets can easily be derived by representing the simulation hierarchy as a tree (see the right-hand side of Fig. 1). For a specific adversary, to every leaf the value 1 is assigned if the corresponding player is noncorrupted, and 0 is assigned if the corresponding player is corrupted. To every inner node, 1 is assigned if and only if more than two-thirds of its children have 1 associated (more than half in the passive model). The considered adversary is tolerated exactly if this procedure assigns 1 to the root node. More formally, the tree corresponds to a circuit with threshold gates, and an adversary is tolerated exactly if the corresponding input vector evaluates to 1.

In this example we have considered a particular simulation tree, and derived the tolerated adversary structure. Deriving and proving the tolerated adversary structure of a simulation is one major goal of this paper. Another goal will be to find such a simulation tree for any given adversary structure.

1.6 Subsequent Work

Subsequently to [HM97], several extensions and improvements for general adversaries were suggested. In [BW98] a more efficient protocol for the passive model is proposed, and the results are formulated in terms of quorum systems. In [CDM98] efficient and modular protocols secure against general adversaries are given for the active and passive model with unconditional and computational security. The efficiency of the protocols for the active model with broadcast is improved in [SS98]. Finally, in [FHM99], a new model with general (nonthreshold) mix-type (active and passive at the same time) adversaries is proposed and tight bounds on the existence of such protocols are given.

1.7 Outline of the Paper

The basic technique for constructing a protocol that tolerates a given adversary structure is to begin with a protocol among a few players and to simulate successively some players by subprotocols among appropriate sets of players. In Section 2 we formalize protocols and adversaries and describe the passive and the active models. In Section 3 we show what it means to simulate a player by a subprotocol and we derive the exact tolerated adversary structures for protocols in which players are simulated by other multiparty protocols. The exact characterization of tolerable adversary sets for both models are presented in Section 4. This is achieved by deriving an appropriate sequence of player simulations from any given admissible adversary structure. In contrast to Section 3, which is concerned with the security of protocols with simulated players, the arguments of Section 4 are purely combinatorial. In Section 5 we show that for some adversary structures, no secure protocols with polynomial efficiency (in the number of players) can exist. Finally, some open problems are mentioned in Section 6.

2 Definitions and Models

Defining and proving the security of multiparty computation protocols is known to be very delicate. In order to be on safe grounds, it is unavoidable to be rather formal in the definitions and proofs. Our definitions are based on Canetti's recent natural and general definitions of security in multiparty computation [Can98b].

2.1 Players, Processors, and Communication

In the literature and also in the previous section, players are assumed to perform two entirely different tasks: On one hand, they provide input and receive output, and on the other hand, they are supposed to perform the operations of the actual protocol. It is necessary to distinguish clearly between these two tasks. Therefore, in the sequel, we refer to a *player* only as the entity that provides input and receives output, and to the associated *processor* as the entity that performs the operations of the protocol. This distinction is important for taking into account the fact that in a general multiparty specification with several input stages, the players' inputs can depend on information obtained during the execution of, but outside of, the protocol (e.g., insider information in a stock-market protocol). The players' computational resources need not be restricted.

A processor can perform operations in a fixed finite field $(\mathcal{F}, +, *)$, can select elements from this field at random, and can communicate with other processors over perfectly authenticated and confidential synchronous channels. The processors are polynomially bounded. In addition to processors associated with players, we also introduce the abstract concept of a *virtual processor*, which offers the same functionality as a processor but only appears in the construction of a protocol. In particular, the trusted party of a specification (or other simulated processors) are virtual processors. Processors are denoted by p_i, where positive indices refer to real processors and negative indices refer to virtual processors.

Formally, a processor can be modeled as a probabilistic Turing Machine, with a (read-only) input tape, a (write-only) output tape, and a (read–write) working tape. The player associated with a processor can write its input tape and can read its output tape. The input and output tapes of virtual processors are not used. Every pair of processors can communicate via a pair of tapes, where one tape is read-only for the first and write-only for the second processor, and the other tape is write-only for the first and read-only for the second processor. All tapes (in particular the communication tapes) are private and authentic, i.e., only the involved processors can read from (or obtain any information about) or write to a tape.

2.2 Variables and Views

We assume a global *variable space* $\mathcal{X}$. A *variable* $x \in \mathcal{X}$ can take on a value from the given finite field $(\mathcal{F}, +, *)$. Every quantity ever generated during a protocol execution, including inputs, local data (e.g., shares), and outputs, is assigned to a variable. For a particular protocol execution each variable takes on only one particular value, i.e., variables are not to be understood in the sense of an imperative programming language but rather as labels for values or, more precisely, as a fixed binding between a name and a value.

The locality or confidentiality of the value assigned to a variable, i.e., the fact that certain variables are seen only by certain processors or sets of processors, is modeled by associating a *view* $\nu(p) \subseteq \mathcal{X}$ with every processor p, capturing the set of variables known to p. The view $\nu(B)$ of a set B of processors is the union of the views of the processors in B. Note that a processor may have full or partial knowledge about a variable although the variable is not in its view. A local variable of a processor is in the global variable space but is only in the view of this processor. Transmitting the value of a variable from one processor to another processor means to include the variable in the latter's view.

2.3 Protocols, Specifications, and Protocol Generators

A *protocol* π among a set P of processors that involves variables from a variable space $\mathcal{X}$ is a sequence $d_1, \ldots, d_l$ of statements. There are four types of *statements*: An *input statement* $\mathit{input}(p_i, x)$ instructs the processor $p_i \in P$ to read a value from its input tape (i.e., from its associated player) and to assign the value to the variable $x \in \mathcal{X}$. A *transmit statement* $\mathit{transmit}(p_1, p_2, x)$ instructs the processor $p_1 \in P$ to send the value of the variable $x \in \mathcal{X}$ to the processor $p_2 \in P$.[4] An *output statement* $\mathit{output}(p, x)$ instructs the processor $p \in P$ to output the value of the variable x to its associated player. Finally, *computation statements* are of one of three forms: A $\mathit{comp}(p, +, x, x_1, x_2)$-statement (a $\mathit{comp}(p, *, x, x_1, x_2)$-statement) instructs the processor p to add (multiply) the values of the variables x_1 and x_2 and to assign the result to the variable x. A $\mathit{comp}(p, ran, x)$-statement instructs the processor p to select an element from the field at random and to assign the value to the variable x.

Assigning a value to a variable (in an input or computation statement) means to define its (global) value and to include it in the processor's view, and is only admissible if no value has previously been assigned. A processor can only use (in a computation, transmit, or output statement) the values of variables that are globally defined and are included in the processor's view.

A *multiparty computation specification* (or simply called specification) formally describes the cooperation to be performed and the processors that give input to, or receive output from, the computation. Intuitively, a specification specifies the cooperation in an ideal environment involving a trusted party. Formally, a specification is a pair (π_0, τ) consisting of a protocol π_0 among a set P_0 of processors, and the name of a virtual processor $\tau \in P_0$. The protocol π_0 of the specification is also called the *ideal protocol*.

In the special case of secure function evaluation, the protocol of the specification first instructs each processor to receive the input from the associated player and to send this value to τ. Then it instructs τ to compute the agreed function and to send the output to every processor. Finally, it instructs each processor to send the output value to its associated player.

A *multiparty protocol generator* G for the set P_G of processors is a function that takes as input a multiparty computation specification (π_0, τ) involving processors from a set P_0 and returns a protocol π for the processors $\big(P_0 \backslash \{\tau\}\big) \cup P_G$. A *statement index function* for a specification (π_0, τ) and protocol π is a strictly monotone function f,

$$f : \{1, \ldots, |\pi_0| + 1\} \to \{1, \ldots, |\pi| + 1\},$$

where $f(1) = 1$ and $f(|\pi_0| + 1) = |\pi| + 1$.

The intuition is that a protocol generator G simulates the virtual trusted processor τ by a multiparty computation protocol among the processors in P_G. Each statement of the ideal protocol π_0 is expanded into a sequence of statements, and all these sequences are concatenated to the resulting protocol π. In order to keep track of which subsequence of π resulted from a given statement d_i of the ideal protocol π_0, the statement index function maps the index i of each statement d_i in π_0 to the index $f(i)$ of the first statement in the corresponding expansion in π, i.e., the ith statement of π_0 "is computed" by the sequence $f(i)$ to $f(i+1) - 1$ of statements of π (since $i = |\pi_0|$ is possible the domain of f includes $|\pi_0| + 1$).

[4] In order to guarantee that every variable has a unique value in a protocol, one could more formally, but equivalently, define a transmit statement as $\mathit{transmit}(p_1, p_2, x_1, \text{“}x_2\text{”})$, which instructs the processor p_1 to send the value of the variable x_1, and p_2 to assign the received value to the variable x_2.

A *BGW multiparty protocol generator* G is a multiparty protocol generator that is constructed using the tools of [BGW88] (see Section 2.7 for more details).

2.4 Structures

A *structure* $\mathcal{Z}$ for the set P of processors is a monotone set of subsets of P, i.e., $\mathcal{Z} \subseteq 2^P$, where all subsets of Z are in $\mathcal{Z}$ if $Z \in \mathcal{Z}$. For a structure $\mathcal{Z}$, $\overline{\mathcal{Z}}$ denotes the *basis* of the structure, i.e., the set of the maximal sets in $\mathcal{Z}$:

$$\overline{\mathcal{Z}} = \left\{ Z \in \mathcal{Z} : \nexists Z' \in \mathcal{Z} : Z \subset Z' \right\}.$$

To *restrict* a structure $\mathcal{Z}$ to the set P' of processors means that all sets in $\mathcal{Z}$ are intersected with P', i.e.,

$$\mathcal{Z}\big|_{P'} = \left\{ Z \cap P' : Z \in \mathcal{Z} \right\}.$$

Note that a restricted monotone structure is still monotone but a restricted basis is not necessarily a basis. (However, we have $\overline{\mathcal{Z}|_{P'}} \subseteq \overline{\mathcal{Z}}\big|_{P'}$). We also use this operator to restrict elements of a structure to a set of processors (i.e., $Z\big|_{P'}$ stands for $Z \cap P'$).

2.5 Adversaries and Definition of Security

Let π be a protocol for the set P of processors. A (static) *passive adversary* A for the protocol π that corrupts the processors in $Z_A \subseteq P$ is a (probabilistic) program (or strategy). After each statement of the protocol π, the passive adversary may read the variables in the views of the corrupted processors (i.e., the variables in $\nu(Z_A)$), and extend its current view by these values.[5] Then it can perform an arbitrary computation on the values in its view and extend its view by the computed values.[6] We do not give a more precise definition of the adversary's view but it is understood that it consists of random variables with a well-defined range. For instance, if the adversary is modeled as a Turing machine, the view consists of the content of all tapes. The complexity of an adversary is not assumed to be polynomial.

A (static) *active adversary* A for the protocol π that corrupts the processors in $Z_A \subseteq P$ is a passive adversary which in addition may stop the corrupted processors and take complete control over their communication tapes. This means that the adversary can read the complete internal state of the corrupted processors and impersonate them in the remaining protocol.

The following definitions of security apply to both passive and active adversaries. For an adversary A, a protocol A*-securely computes* a specification if, whatever A does in the protocol, the same effect could be achieved by an adversary (with a modified strategy, but with similar costs) in the ideal protocol of the specification.

Formally, for an adversary A and a specification (π_0, τ) for the set P_0 of processors, the protocol π A-securely computes the specification (π_0, τ) if there exists a statement index function $f_\pi : \{1, \dots, |\pi_0| +$

[5]It may at first appear to be sufficient to assume that the adversary reads the variables of all corrupted processors only at the end of the protocol. However, in our construction a protocol may consist of several intertwined protocols and values appearing in one of them could be of use in selecting inputs corresponding to another protocol; therefore it must be tolerated that the adversary reads the variables after every statement. This corresponds to the notion of "on-line security" in [Can98b].

[6]The values in the adversary's view need neither be elements of the finite field nor be assigned to variables of the global variable space. However, such a restriction could be made without loss of generality.

$1\} \to \{1, \ldots, |\pi| + 1\}$ and an adversary A_0 for the ideal protocol π_0 with[7] $Z_{A_0} = Z_A\big|_{P_0 \setminus \{\tau\}}$ such that for all inputs and for every $i = 1, \ldots, |\pi_0| + 1$ the joint distribution of A_0's view and the views $\nu_i(p)$ of all noncorrupted processors $p \in \left(P_0 \setminus \{\tau\} \setminus Z_{A_0}\right)$ before the ith statement of the ideal protocol π_0 (with the adversary A_0 present) is equal to the joint distribution of A's view and the views $\nu_{f(i)}(p)$ of all noncorrupted processors $p \in \left(P_0 \setminus \{\tau\} \setminus Z_{A_0}\right)$ before the $f(i)$th statement of the real protocol π (with the adversary A present). Moreover, the complexity of A_0 must be polynomial in the complexity of A. This corresponds to the definition of on-line security of [Can98b]. The adversary A_0 can be seen as a kind of simulator and is called the *ideal adversary* of A.

For the special case of secure function evaluation, the only effect that an adversary A can achieve in a protocol that A-securely computes this specification corresponds to a modification of the inputs and outputs of the corrupted processors in the ideal protocol (which of course cannot be prevented).

For a structure $\mathcal{Z}$ and a specification (π_0, τ), a protocol π *$\mathcal{Z}$-securely computes* the specification (π_0, τ) if, for every adversary A with $Z_A \in \mathcal{Z}$, the protocol π A-securely computes the specification (π_0, τ). Whenever the specification is clear from the context, we also say that a protocol *tolerates* an adversary A (a structure $\mathcal{Z}$) instead of saying that the protocol A-securely ($\mathcal{Z}$-securely) computes the specification.

A protocol generator G for the set P of processors is *A-secure* (or *tolerates* A) for a given adversary A if, for every specification, the protocol that results by applying the generator to this specification A-securely computes the specification. For a structure $\mathcal{Z} \subseteq 2^P$, a protocol generator G for the set P of processors is *$\mathcal{Z}$-secure* (or *tolerates* $\mathcal{Z}$) if, for every adversary A with $Z_A\big|_P \in \mathcal{Z}$, the protocol generator is A-secure.[8]

2.6 Models

We consider the same two models as in [BGW88]. In the *passive model*, only a passive adversary may be present. In the *active model*, only an active adversary may be present. In both models, we assume reliable synchronous secure channels between every pair of processors but we do not assume a broadcast channel. The basic protocols of [BGW88] can be realized without broadcast or, more precisely, by simulating it with a protocol among the sender and the receivers of the broadcast (e.g., [LSP82], [FM88], [BDDS92], and [FM97]).

In both models, we only consider a static adversary, but the protocols are also secure against an adaptive adversary. For the sake of simplicity, the proofs are not extended to capture the additional power of adaptive adversaries.

2.7 BGW Protocol Generators

We use a particular BGW protocol generator for each model: G^{p3} denotes the three-party BGW protocol generator of [BGW88] for the set $P_{G^{\mathsf{p3}}} = \{p_1, p_2, p_3\}$ of processors for the passive model, tolerating all passive adversaries that may corrupt one single processor, and G^{a4} denotes the four-party BGW protocol

[7] It is necessary to explicitly exclude τ because it is possible that τ occurs in π, and even if A may corrupt τ (which thus is a simulating processor), it cannot be tolerated that A_0 corrupts τ (which is the trusted party of the specification). At this point, this technicality appears to be pedantic, but in later recursive constructions it will be necessary. Note that if τ does not occur in π, then $Z_A\big|_{P_0 \setminus \{\tau\}} = Z_A\big|_{P_0}$.

[8] The intuitive condition $Z_A \in \mathcal{Z}$ is too restrictive, because this would not include adversaries that corrupt processors of the specification.

generator for the set $P_{G^{\mathsf{a4}}} = \{p_1, p_2, p_3, p_4\}$ of processors of [BGW88] in the active model, tolerating all active adversaries that may corrupt one single processor.

The protocol generators G^{p3} and G^{a4} are realized as follows: For a given specification (π_0, τ), they scan π_0 statement by statement and generate a new protocol in which each statement involving τ is replaced by a statement sequence.

In the passive model, G^{p3} for the set $P_{G^{\mathsf{p3}}} = \{p_1, p_2, p_3\}$ of processors is defined as follows: All statements of π_0 that do not involve τ are left unchanged. Every statement *transmit*(p, τ, x) is replaced by a secret sharing protocol in which p is the dealer who shares the variable x among the processors p_1, p_2, and p_3 such that any two of them can reconstruct the secret. Every statement *transmit*(τ, p, x) is replaced by the subprotocol to reconstruct the secret, in which the processors p_1, p_2, and p_3 send their shares to p who then interpolates the secret. Every statement *comp*$(\tau, +, x, x_1, x_2)$ is replaced by the three statements that instruct the processors p_1, p_2, and p_3 to add their shares of x_1 and of x_2 and to assign the result to the variable of their share of x. Every statement *comp*$(\tau, *, x, x_1, x_2)$ is replaced by the multiplication protocol of [BGW88] (improved by [GRR98]) that multiplies the shared variables x_1 and x_2 and assigns the resulting shares to the variables of the shares of x.

A statement *comp*(τ, ran, x) is first replaced by a short sequence of statements, still involving τ, but not involving a statement of the form *comp*$(\tau, ran, \dots)$, and then, this sequence is replaced by a sequence without need for the trusted party τ by using the techniques described above. In the first step, each processor p_1, p_2, and p_3 selects a random number (i.e., *comp*(p_i, ran, x_i) for $i = 1, 2, 3$), then each one sends his number to the trusted party τ (i.e., *transmit*(p_i, τ, x_i)), who then computes the random value x as the sum of x_1, x_2, and x_3 (i.e., *comp*$(\tau, +, x', x_1, x_2)$, *comp*$(\tau, +, x, x', x_3)$). It is clear that if at least one of the players p_1, p_2, or p_3 is honest, then x is a uniformly selected secure random number.

In the active model, G^{a4} is constructed similarly. Instead of the secret sharing protocol, the verifiable secret sharing protocol of [BGW88] is used. Moreover, reconstruction involves error correction. As multiplication protocol we use the protocol that robustly multiplies two shared values, as described in [BGW88] and [GRR98]. The protocol to select jointly a random field element (as described above) uses verifiable secret sharing.

The protocol generators G^{p3} and G^{a4} are indeed $\mathcal{Z}$-secure for $\mathcal{Z} = \big\{\{p_1\}, \{p_2\}, \{p_3\}\big\}$ (passive model, G^{p3}) or $\mathcal{Z} = \big\{\{p_1\}, \{p_2\}, \{p_3\}, \{p_4\}\big\}$ (active model, G^{a4}), respectively. The security is claimed in [BGW88], but is not formally proven. It is outside the scope of this paper to fill this gap, but such a proof is in preparation [Can98a]. In the sequel, we assume that G^{p3} and G^{a4} are secure.

3 Processor Simulation

In this section we introduce the technique of simulating (virtual) processors by other processors.[9] In a first step (Section 3.2), virtual processors are simply *renamed* using a *processor mapping*, i.e., one (virtual) processor plays for one or several virtual processors. In a second step (Section 3.3), virtual processors are *simulated* by a set of (virtual) processors, i.e., the simulating processors perform all operations of the simulated virtual processor by a multiparty computation.

[9]The idea of simulating a single processor by a subprotocol was used in [Cha89] for a different purpose.

3.1 Definitions

Let P and P' be sets of processors. A *processor mapping* σ,

$$\sigma : P \to P',$$

is a surjective function from P onto P'.[10] The definition of a processor mapping σ is extended to the following domains: For a protocol π, the mapped protocol $\sigma(\pi)$ (or, equivalently, $\sigma\pi$) is the same protocol, where in each statement all involved processors are replaced by the corresponding mapped processors (if processors that are not in P are involved in a statement, then these processors are not replaced). For a specification (π_0, τ) with $\tau \notin P$, the mapped specification is the specification with the mapped protocol, i.e., $\sigma(\pi_0, \tau) = (\sigma\pi_0, \tau)$. Note that $\sigma(\pi_0, \tau)$ stands for $\sigma\big((\pi_0, \tau)\big)$.

The *inverse processor mapping* σ^{-1} of a processor mapping σ is defined by

$$\sigma^{-1}:\ P' \to 2^P,\ p' \mapsto \big\{p \in P\colon \sigma(p) = p'\big\}\,.$$

If the processor mapping σ is bijective, then the function value of the inverse processor mapping σ^{-1} is sometimes interpreted as a single processor (instead of a set that contains a single processor).[11] Also, we define the mapping of a set of processors to be the set of the mapped processors, and the inverse mapping of a set B of processors to be the union of the sets of the inverse mappings applied to the processors in B (i.e., $\sigma(B) = \bigcup_{p \in B}\big\{\sigma(p)\big\}$ and $\sigma^{-1}(B) = \bigcup_{p \in B} \sigma^{-1}(p)$).

In the following we give definitions for applying processor mappings to adversary structures and to protocol generators. These definitions are appropriate in the sense that if a protocol (generator) tolerates an adversary structure, then the mapped protocol (generator) will tolerate the mapped adversary structure. This will be proven in the next section.

For a structure $\mathcal{Z}$ for the set P of processors and a processor mapping $\sigma : P \to P'$, the mapped structure is

$$\sigma(\mathcal{Z}) = \big\{Z \subseteq P'\colon \sigma^{-1}(Z) \in \mathcal{Z}\big\}\,,$$

i.e., a set Z is in $\sigma(\mathcal{Z})$ if the set of all processors mapped to a processor in Z is in $\mathcal{Z}$. For a protocol generator G for the set P of processors and a processor mapping $\sigma : P \to P'$, the mapped protocol generator $\sigma(G)$ is a protocol generator that, applied to a specification (π_0, τ), simulates the trusted party τ by the processors in P' (instead of P). In order to prevent syntactical collisions with the names of the processors in P and of those appearing in π_0, we first rename the processors appearing in π_0 to some new processor names,[12] then apply the original protocol generator G, then apply the processor mapping σ, and finally rename the previously renamed processors back to their original names. More formally, when given (π_0, τ) where π_0 involves the set P_0 of processors, $\sigma(G)$ first applies an arbitrary bijective

[10] In the application of processor mappings, parentheses may be omitted whenever the precedence rules allow it. As usual, function application (in particular a processor mapping) is right-associative and has higher precedence than any two-adic operator. For any two processor mappings σ_1 and σ_2, for an arbitrary two-adic operator $\diamond$, and for any x_1 and x_2 we have $\sigma_1\sigma_2 x_1 \diamond x_2 = \sigma_1(\sigma_2(x_1)) \diamond x_2$.

[11] Generally, the inverse of a processor mapping is not a processor mapping. However, the inverse of a *bijective* processor mapping can be considered (and will be considered) as a processor mapping.

[12] That is, processor names that did not yet appear anywhere, neither in the protocol nor in the protocol generator nor in the mapping.

processor mapping[13] $\rho : (P_0 \setminus \tau) \to \overline{P}$, where $\overline{P}$ is a set of new processor names, to the specification, then applies the protocol generator G to this modified protocol specification, further applies the original processor mapping σ and finally applies the inverse processor mapping ρ^{-1} to the resulting protocol. Formally,

$$\sigma G = \sigma(G) = \Big((\pi_0, \tau) \mapsto \rho^{-1}\big(\sigma(G(\rho\pi_0, \tau))\big) \Big)$$

for an appropriate bijective processor mapping ρ. Note that σG does not depend on the choice of ρ.

Consider a multiparty protocol π among the set P of processors and a protocol generator G for the set P_G of processors. To *simulate* a virtual processor $p \in P$ in π applying the protocol generator G means to consider this processor p as a trusted party and to have this party simulated by a subprotocol among the processors in P_G, according to G. More precisely, the specification (π, p) is used as input for the protocol generator G. To *simultaneously simulate* the processors $p_{r_1}, \ldots, p_{r_k} \in P$ in π using the protocol generators $G_1, \ldots, G_k$ for the processor sets $P_1, \ldots, P_k$, respectively, is defined as follows: First consider k arbitrary bijective processor mappings $\sigma_i : P_i \to \overline{P}_i$ (for $i = 1, \ldots, k$), where $\overline{P}_1, \ldots, \overline{P}_k$ are pairwise disjoint sets of new processor names. Then the resulting protocol is

$$\sigma_k^{-1} \cdots \sigma_1^{-1} \Big((\sigma_k G_k) \big(\cdots (\sigma_2 G_2) \big((\sigma_1 G_1)(\pi, p_{r_1}), p_{r_2} \big) \cdots, p_{r_k} \big) \Big),$$

and does not depend on the choices for $\sigma_1, \ldots, \sigma_k$.

3.2 Renaming Processors

It is trivial that by renaming processors in a protocol, the tolerated adversary structure is the same with the identically renamed processors. More precisely, security of a protocol is defined with respect to a specification, and the security of the renamed protocol is with respect to the renamed specification. Furthermore, when several processors are renamed to the same processor p (i.e., p "plays" the role of several processors), then a subset Z of the processors that contains p is tolerated in the renamed protocol if and only if the set of all the renamed processors and all the processors in $Z \setminus \{p\}$ is tolerated in the original protocol. A subset Z with $p \notin Z$ is tolerated if Z is tolerated in the original protocol. This naturally extends the "partition lemma" of [CK89].

Lemma 1 *Given a protocol π for the set P of processors that $\mathcal{Z}$-securely computes the specification (π_0, τ), and some processor mapping σ, then $\sigma(\pi)$ is a protocol for the set $\sigma(P)$ of processors that $\sigma(\mathcal{Z})$-securely computes the specification $\sigma(\pi_0, \tau)$.*

Proof. We have to show that for every adversary A' for the mapped protocol $\sigma(\pi)$, with $Z_{A'} \in \sigma(\mathcal{Z})$, the protocol $\sigma(\pi)$ A'-securely computes the mapped specification $\sigma(\pi_0, \tau)$. Figure 2 illustrates the procedure for constructing an ideal adversary A_0' for the mapped specification $\sigma(\pi_0, \tau)$ from a given adversary A' for $\sigma\pi$. We begin with the adversary A' for the mapped protocol $\sigma\pi$, construct an adversary A for the original protocol π, and show that A is tolerated in the protocol π. Thus, by the definition of security of a protocol, there exists an ideal adversary A_0 for the protocol π_0 of the specification. Then we use A_0 to construct an adversary A_0' for the mapped specification $\sigma(\pi_0, \tau)$, and we prove that this adversary is an ideal adversary of the original adversary A'.

[13]This corresponds to alpha renaming in the context of lambda calculus and is a purely technical step. Note that the name of the trusted party must not be mapped.

$$\begin{tikzcd}
 & \text{ideal} & \text{real} \\
\text{mapped} & A_0' \text{ for } \sigma\pi_0 & A' \text{ for } \sigma\pi \arrow[d, "\sigma^{-1}"] \\
\text{unmapped} & A_0 \text{ for } \pi_0 \arrow[u, "\sigma"] & A \text{ for } \pi \arrow[l]
\end{tikzcd}$$

Figure 2: Construction of the ideal adversary A_0' for a given adversary A'.

Consider an arbitrary adversary A' for $\sigma\pi$ with $Z_{A'} \in \sigma(\mathcal{Z})$. We define A to be the adversary for π with $Z_A = \sigma^{-1}(Z_{A'})$ and with the same strategy as A', except that whenever A' reads from, or writes to, the tape of a corrupted processor $p' \in Z_{A'}$, then A accesses the tape of the corresponding[14] processor $p \in \sigma^{-1}(p')$ in the same manner as A'. By the definition of processor mappings for structures, $Z_{A'} \in \sigma(\mathcal{Z})$ implies that $Z_A \in \mathcal{Z}$. Hence, by what it means for a protocol to be A-secure, there exists a statement index function $f_\pi : \{1, \ldots, |\pi_0| + 1\} \to \{1, \ldots, |\pi| + 1\}$ and an adversary A_0 for π_0 with $Z_{A_0} = Z_A\big|_{P_0 \setminus \{\tau\}}$ (where P_0 is the set of processors of the ideal protocol π_0) such that for every $i = 1, \ldots, |\pi_0| + 1$ the joint distribution of the view of the adversary A_0 and the views $\nu_i(p)$ of all noncorrupted processors $p \in \big(P_0 \setminus \{\tau\} \setminus Z_{A_0}\big)$ before the ith statement of the ideal protocol π_0 (with the adversary A_0 present) is equal to the joint distribution of the view of the adversary A and the views $\nu_i(p)$ of all noncorrupted processors $p \in \big(P_0 \setminus \{\tau\} \setminus Z_{A_0}\big)$ before the $f_\pi(i)$th statement of the real protocol π (with the adversary A present). Let the statement index function $f_{\sigma\pi}$ for the mapped protocol be equal to that of the unmapped protocol, i.e., $f_{\sigma\pi} = f_\pi$, and let A_0' be the adversary for the mapped ideal protocol $\sigma\pi_0$ with $Z_{A_0'} = \sigma(Z_{A_0}) = \sigma\big(Z_A\big|_{P_0 \setminus \{\tau\}}\big) = \sigma\big(\sigma^{-1}(Z_{A'})\big|_{P_0 \setminus \{\tau\}}\big) = Z_{A'}\big|_{\sigma(P_0) \setminus \{\tau\}}$. The strategy of A_0' is the same as the strategy of the adversary A_0, except that whenever A_0 reads from, or writes to, the tape of a corrupted processor $p \in Z_{A_0}$, then A_0' accesses the tape of the processor $\sigma(p)$.

Clearly, for each $i = 1, \ldots, |\sigma\pi_0| + 1$, the joint distribution of the view of the adversary A_0' and the views $\nu_i(p)$ of all noncorrupted processors $p \in \big(\sigma(P_0) \setminus \{\tau\} \setminus Z_{A_0'}\big)$ before the ith statement of the mapped ideal protocol $\sigma\pi_0$ (with the adversary A_0' present) is equal to the joint distribution of the view of the adversary A' and the views $\nu_{f_{\sigma\pi}(i)}(p)$ of all noncorrupted processors $p \in \big(\sigma(P_0) \setminus \{\tau\} \setminus Z_{A_0'}\big)$ before the $f_{\sigma\pi}(i)$th statement of the mapped real protocol $\sigma\pi$ (with the adversary A' present). $\square$

If the structure $\mathcal{Z}$ is maximal for the protocol π (i.e., for every $Z \subseteq P$ with $Z \notin \mathcal{Z}$ there exists a specification (π_0, τ) and an adversary A with $Z_A \subseteq Z$, such that π does not A-securely compute (π_0, τ)), then $\sigma(\mathcal{Z})$ is also maximal for $\sigma(\pi)$.

The corollary below follows immediately from Lemma 1 and from the definition of processor mappings for protocol generators.

Corollary 1 *Given a $\mathcal{Z}$-secure protocol generator G for the set P_G of processors, and given some processor mapping σ, then $\sigma(G)$ is a $\sigma(\mathcal{Z})$-secure protocol generator for the set $\sigma(P_G)$ of processors.*

[14]If the mapping is not bijective, then for constructing the adversary A one must consult the unmapped protocol π to determine which processor's tape needs to be accessed.

3.3 Simulating a Single Processor

Consider a protocol π for the set P of processors, a processor $p \in P$, and a protocol generator G' for the set P' of processors, and assume that π is $\mathcal{Z}$-secure for an adversary structure $\mathcal{Z}$ and G' is $\mathcal{Z}'$-secure for an appropriate adversary structure $\mathcal{Z}'$. Let p be simulated in π by G', and let π^* denote the resulting protocol. A set Z of processors is tolerated in π^* if the set Z is tolerated in π (i.e., $Z|_P \in \mathcal{Z}$) and Z is tolerated by G' (i.e $Z|_{P'} \in \mathcal{Z}'$). Even if Z is not tolerated by G', but instead π tolerates that p is corrupted in addition to the processors in Z (i.e., $\big(Z|_P \cup \{p\}\big) \in \mathcal{Z}$), then Z is nevertheless tolerated in the resulting protocol π^*. This is formally stated and proved below.

Lemma 2 *Consider a specification (π_0, τ) for the set P_0 of processors, a protocol π for the set P of processors that $\mathcal{Z}$-securely computes (π_0, τ) for some adversary structure $\mathcal{Z} \subseteq 2^P$, and a BGW protocol generator G' for the set P' of processors (where $P' \cap P = \emptyset$) that is $\mathcal{Z}'$-secure for some adversary structure $\mathcal{Z}' \subseteq 2^{P'}$. Simulating a processor $p \in \big(P \backslash \{\tau\} \backslash P_0\big)$ in π by applying the protocol generator G' results in a protocol π^* for the set P^* of processors that $\mathcal{Z}^*$-securely computes the specification (π_0, τ) where*

$$\begin{aligned} P^* &= \big(P \backslash \{p\}\big) \cup P' \\ \mathcal{Z}^* &= \Big\{ Z \subseteq P^* \colon \Big(\big(Z|_P \cup \{p\}\big) \in \mathcal{Z} \Big) \vee \Big(Z|_P \in \mathcal{Z} \wedge Z|_{P'} \in \mathcal{Z}' \Big) \Big\}. \end{aligned}$$

Proof. Consider an arbitrary adversary A^* for the protocol π^* with $Z_{A^*} \in \mathcal{Z}^*$, and a statement index function $f' : \{1, \dots, |\pi| + 1\} \to \{1, \dots, |\pi^*| + 1\}$. We construct a statement index function $f^* : \{1, \dots, |\pi_0| + 1\} \to \{1, \dots, |\pi^*| + 1\}$ and an ideal adversary A_0 of the adversary A^* where $Z_{A_0} = Z_{A^*}|_{P_0 \backslash \{\tau\}}$. We distinguish between two cases: in the first case, we assume that $\big(Z_{A^*}|_P \cup \{p\}\big) \in \mathcal{Z}$, and in the second case we assume that $Z_{A^*}|_P \in \mathcal{Z} \wedge Z_{A^*}|_{P'} \in \mathcal{Z}'$.

First, assume that $\big(Z_{A^*}|_P \cup \{p\}\big) \in \mathcal{Z}$. We define A to be the adversary for the protocol π with $Z_A = Z_{A^*}|_P \cup \{p\}$ with the following strategy: Without loss of generality, let $P' = \{p_1, \dots, p_m\}$. First, A locally initializes $|P'|$ virtual processors $\widetilde{p}_1, \dots, \widetilde{p}_m$, one for each processor in P', and an additional virtual processor $\widetilde{p}_0$ for simulating the behavior of a noncorrupted sender or receiver in the protocol π^* ($\widetilde{p}_0$ is re-initialized after every such use), and assigns an empty view to each of them. For every $i = 1, \dots, |\pi| + 1$, A performs the following steps for the ith statement d_i of π:

- If the statement d_i does not involve p, A performs the same steps that A^* would perform.
- If d_i is a *comp*-statement for p, then A executes the sequence $f'(i), \dots, \big(f'(i+1) - 1\big)$ of statements of π^*, where each processor $p_i \in P'$ is relabeled to the virtual processor $\widetilde{p}_i$.[15] After each statement of this sequence, A performs the same steps that A^* would perform (modified such that it accesses the tapes of $\widetilde{p}_i$ instead of p_i).
- If d_i is a *transmit*(p, p_j, x)-statement, then A executes the sequence $f'(i), \dots, f'(i+1) - 1$ of statements of π^*, where each processor $p_i \in P'$ is relabeled to the virtual processor $\widetilde{p}_i$, and the receiving processor p_j is renamed to $\widetilde{p}_0$. After each statement of this sequence, A performs the same steps that A^* would perform (where A accesses the tapes of $\widetilde{p}_i$ instead of p_i, and of $\widetilde{p}_0$ instead of p_j). At the end of the sequence, A reads the value of the variable x in the view of $\widetilde{p}_0$ and sends this value to p_j. If the adversary A^* is passive (and hence A also is passive), then this value corresponds to the value that p would send to p_j.

[15]More precisely, a processor mapping σ that maps $p_i \mapsto \widetilde{p}_i$ $\big(1 \le i \le m\big)$ is applied to the sequence of statements.

- If d_i is a *transmit*(p_j, p, x)-statement, then A first reads the value of x and puts this value into the view of $\widetilde{p}_0$, then executes the sequence $f'(i), \ldots, f'(i+1) - 1$ of statements of π^*, where each processor $p_i \in P'$ is first relabeled to the virtual processor $\widetilde{p}_i$, and the sending processor p_j is renamed to $\widetilde{p}_0$.[16] After each statement of this sequence, A performs the same steps that A^* would perform (modified as above).

The described adversary A is tolerated in π because $Z_A \in \mathcal{Z}$. Thus there exists an ideal adversary A_0 of A with $Z_{A_0} = Z_A\big|_{P_0 \setminus \{\tau\}}$. Clearly, this is also an ideal adversary for A'.

Second, assume that $Z_{A^*}\big|_P \in \mathcal{Z} \wedge Z_{A^*}\big|_{P'} \in \mathcal{Z}'$. A^* is tolerated by the protocol generator G' (because $Z_{A^*}\big|_{P'} \in \mathcal{Z}'$); thus by considering (π, p) as the specification of π^*, there exists an ideal adversary A of A^* for the protocol π with $Z_A = Z_{A^*}\big|_P$. Because $Z_A \in \mathcal{Z}$ there exists an ideal adversary A_0 of A for the ideal protocol π_0 with $Z_{A_0} = Z_A\big|_{P_0 \setminus \{\tau\}}$. One can easily verify that A_0 is also an ideal adversary of A^*. □

3.4 General Simulation of Processors

In this section we consider the simultaneous simulation of several processors with completely general (possibly overlapping) sets of simulating processors. In a protocol resulting from such a simulation, an adversary is tolerated if every corrupted nonsimulated processor is tolerated in the original protocol and, in addition, for every simulated processor, either the adversary is tolerated in the corresponding subprotocol (more precisely, by the corresponding protocol generator), or this processor is tolerated to be (additionally) corrupted in the original protocol. This is formally stated and proved below.

Theorem 1 *Let π be a protocol among the set P of processors that $\mathcal{Z}$-securely compute a specification (π_0, τ), and let $G_1, \ldots, G_k$ be $\mathcal{Z}_1$-, $\ldots$, $\mathcal{Z}_k$-secure BGW protocol generators for the processor sets $P_1, \ldots, P_k$, respectively.*[17] *Assume that in π the k processors $p_{r_1}, \ldots, p_{r_k} \in P$ are simultaneously simulated by subprotocols applying the protocol generators $G_1, \ldots, G_k$, respectively. Then the resulting multiparty protocol π^* is for the set P^* of processors and $\mathcal{Z}^*$-securely computes the specification (π_0, τ), where*

$$P^* = (P \setminus R) \cup \bigcup_{i=1}^{k} P_i,$$
$$\mathcal{Z}^* = \left\{ Z \subseteq P^* : \left(Z\big|_{P \setminus R} \cup \left\{ p_{r_i} \in R : Z\big|_{P_i} \notin \mathcal{Z}_i \right\} \right) \in \mathcal{Z} \right\},$$

and $R = \{p_{r_1}, \ldots, p_{r_k}\}$ is the set of replaced processors.

[16] Here we assume that the statements in the sequence $f(i), \ldots, f(i+1) - 1$ of π^* require p_j to know only the value of the variable x but of no other variables. This is the case for example for BGW protocol generators. For most other protocol generators (e.g., [RB89]), the proof can be adapted such that additional information (e.g., check vectors) are associated with each variable. However, one can construct artificial protocol generators for which this lemma does not hold. For example, consider the protocol generator that is almost identical to the BGW protocol generators with the only exception that a *transmit*(p_j, p, x) is translated into a secret sharing protocol for *all* variables in the view $\nu(p_j)$ of p_j (and not only for x). This protocol generator is still secure, but an untolerated adversary A^* learns the whole view of p_j, which cannot be simulated in the protocol π.

[17] Note that $G_1, \ldots, G_k$ are generally mapped versions of G^{p3} or G^{a4}.

Proof. According to the definition of simultaneous simulation,

$$\pi^* = \sigma_k^{-1} \cdots \sigma_1^{-1}\Big(\big(\sigma_k G_k\big)\big(\cdots(\sigma_2 G_2)((\sigma_1 G_1)(\pi, p_{r_1}), p_{r_2}) \cdots, p_{r_k}\big)\Big)$$

for some bijective processor mappings $\sigma_1 : P_1 \to \overline{P}_1, \ldots, \sigma_k : P_k \to \overline{P}_k$, where $\overline{P}_1, \ldots, \overline{P}_k$ are pairwise disjoint sets of new processor names. According to Corollary 1, $\sigma_1 G_1, \ldots, \sigma_k G_k$ are BGW protocol generators for the sets $\sigma_1 P_1, \ldots, \sigma_k P_k$ of processors that are $\sigma_1 \mathcal{Z}_1, \ldots, \sigma_k \mathcal{Z}_k$ secure, respectively.

These protocol generators are applied subsequently, where after applying the ith generator the set of processors is denoted by $P^{(i)}$ and the tolerated adversary structure is denoted by $\mathcal{Z}^{(i)}$. In the following, some technical transformations of $\mathcal{Z}^{(i)}$ may at first glance appear to be unmotivated.

Applying Lemma 2, $(\sigma_1 G_1)(\pi, p_{r_1})$ is a protocol for the set $P^{(1)}$ of processors tolerating $\mathcal{Z}^{(1)}$, where

$$\begin{aligned}
P^{(1)} &= \big(P \backslash \{p_{r_1}\}\big) \,\cup\, \sigma_1 P_1 \\
\mathcal{Z}^{(1)} &= \left\{ Z \subseteq P^{(1)} \colon \begin{array}{l} \Big(\big(Z|_P \cup \{p_{r_1}\}\big) \in \mathcal{Z}\Big) \\ \bigvee \ \Big(Z|_P \in \mathcal{Z} \wedge Z|_{\sigma_1 P_1} \in \sigma_1 \mathcal{Z}_1\Big) \end{array} \right\} \\
&= \Big\{ Z \subseteq P^{(1)} \colon \Big(Z|_P \cup \Big\{p_{r_i} \in \{p_{r_1}\} \colon Z|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i\Big\}\Big) \in \mathcal{Z} \Big\}. \qquad (1)
\end{aligned}$$

Furthermore, $\big(\sigma_2 G_2\big)\big((\sigma_1 G_1)(\pi, p_{r_1}), p_{r_2}\big)$ is a protocol for the set $P^{(2)}$ of processors tolerating $\mathcal{Z}^{(2)}$, where

$$\begin{aligned}
P^{(2)} &= \big(P^{(1)} \backslash \{p_{r_2}\}\big) \,\cup\, \sigma_2 P_2 = \big(P \backslash \{p_{r_1} \cup p_{r_2}\}\big) \cup \sigma_1 P_1 \cup \sigma_2 P_2 \\
\mathcal{Z}^{(2)} &= \left\{ Z \subseteq P^{(2)} \colon \begin{array}{l} \Big(\big(Z|_{P^{(1)}} \cup \{p_{r_2}\}\big) \in \mathcal{Z}^{(1)}\Big) \\ \bigvee \ \Big(Z|_{P^{(1)}} \in \mathcal{Z}^{(1)} \wedge Z|_{\sigma_2 P_2} \in \sigma_2 \mathcal{Z}_2\Big) \end{array} \right\} \\
&= \Big\{ \ Z \subseteq P^{(2)} \colon \underbrace{\Big(Z|_{P^{(1)}} \cup \Big\{p_{r_i} \in \{p_{r_2}\} \colon Z|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i\Big\}\Big)}_{T} \in \mathcal{Z}^{(1)} \ \Big\}.
\end{aligned}$$

We now replace $\mathcal{Z}^{(1)}$ in the above equation by using (1). Let T be the underbraced term.

$$\mathcal{Z}^{(2)} = \Big\{ Z \subseteq P^{(2)} \colon \Big(T|_P \ \cup \Big\{p_{r_i} \in \{p_{r_1}\} \colon T|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i\Big\}\Big) \in \mathcal{Z} \Big\}.$$

We have

$$\begin{aligned}
T|_P &= \Big(Z|_{P^{(1)}} \cup \Big\{p_{r_i} \in \{p_{r_2}\} \colon Z|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i\Big\}\Big)\Big|_P \\
&= \big(Z|_{P^{(1)}}\big)\big|_P \ \cup \ \Big\{p_{r_i} \in \{p_{r_2}\} \colon Z|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i\Big\}\Big|_P \\
&= Z|_P \ \cup \ \Big\{p_{r_i} \in \{p_{r_2}\} \colon Z|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i\Big\}
\end{aligned}$$

and

$$\begin{aligned}
T|_{\sigma_i P_i} &= \Big(Z|_{P^{(1)}} \cup \Big\{p_{r_i} \in \{p_{r_2}\} \colon Z|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i\Big\}\Big)\Big|_{\sigma_i P_i} \\
&= \big(Z|_{P^{(1)}}\big)\big|_{\sigma_i P_i} \ \cup \ \Big\{p_{r_i} \in \{p_{r_2}\} \colon Z|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i\Big\}\Big|_{\sigma_i P_i} \\
&= Z|_{\sigma_i P_i} \ \cup \ \emptyset \ = \ Z|_{\sigma_i P_i}.
\end{aligned}$$

This gives

$$\begin{aligned}
\mathcal{Z}^{(2)} &= \left\{ Z \subseteq P^{(2)} \colon \left(\begin{array}{l} \Big(Z\big|_P \cup \Big\{ p_{r_i} \in \{p_{r_2}\} \colon Z\big|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i \Big\} \Big) \\ \cup \Big\{ p_{r_i} \in \{p_{r_1}\} \colon Z\big|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i \Big\} \end{array} \right) \in \mathcal{Z} \right\} \\
&= \Big\{ Z \subseteq P^{(2)} \colon \Big(Z\big|_P \cup \Big\{ p_{r_i} \in \{p_{r_1}, p_{r_2}\} \colon Z\big|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i \Big\} \Big) \in \mathcal{Z} \Big\} .
\end{aligned}$$

Repeating this step k times yields the set $P^{(k)}$ of processors and the tolerated structure $\mathcal{Z}^{(k)}$ of the protocol $(\sigma_k G_k)(\cdots(\sigma_2 G_2)((\sigma_1 G_1)(\pi, p_{r_1}), p_{r_2})\cdots, p_{r_k})$:

$$\begin{aligned}
P^{(k)} &= \big(P \backslash R\big) \cup \sigma_1 P_1 \cup \cdots \cup \sigma_k P_k \\
\mathcal{Z}^{(k)} &= \Big\{ Z \subseteq P^{(k)} \colon \Big(Z\big|_P \cup \Big\{ p_{r_i} \in R \colon Z\big|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i \Big\} \Big) \in \mathcal{Z} \Big\} .
\end{aligned}$$

Finally, we apply the inverse processor mappings $\sigma_1^{-1}, \ldots, \sigma_k^{-1}$. Let $\vartheta = \sigma_k^{-1} \cdots \sigma_1^{-1}$. Because $\sigma_1^{-1}, \ldots, \sigma_k^{-1}$ are bijective and have pairwise disjoint domains, all function values of ϑ are sets with a single processor and are considered as those processors (rather than as sets). Also, ϑ must be extended to be the identity function for the processors in $P \backslash R$, since it will be applied to the previously constructed protocol among the set $P^{(k)}$ of processors. The resulting protocol π^* for the set P^* of processors tolerates the structure $\mathcal{Z}^*$, where

$$\begin{aligned}
P^* &= \vartheta P^{(k)} \\
&= \vartheta \Big(\big(P \backslash R\big) \cup \sigma_1 P_1 \cup \cdots \cup \sigma_k P_k \Big) \\
&= \big(P \backslash R\big) \cup \sigma_1^{-1} \sigma_1 P_1 \cup \cdots \cup \sigma_k^{-1} \sigma_k P_k \\
&= \big(P \backslash R\big) \cup P_1 \cup \cdots \cup P_k \\
&= \big(P \backslash R\big) \cup \bigcup_{i=1}^{k} P_i \\
\mathcal{Z}^* &= \vartheta \mathcal{Z}^{(k)} \\
&= \Big\{ Z \subseteq \vartheta P^{(k)} \colon \vartheta^{-1}\big(Z\big) \in \mathcal{Z}^{(k)} \Big\} \\
&= \Big\{ Z \subseteq P^* \colon \Big(\vartheta^{-1}(Z)\big|_P \cup \Big\{ p_{r_i} \in R \colon \vartheta^{-1}(Z)\big|_{\sigma_i P_i} \notin \sigma_i \mathcal{Z}_i \Big\} \Big) \in \mathcal{Z} \Big\} .
\end{aligned}$$

Due to the definition of ϑ, we have $\vartheta^{-1}(Z)\big|_P = Z\big|_{P \backslash R}$. Since the sets $\overline{P}_i$ are pairwise disjoint we have $\vartheta^{-1}(Z)\big|_{\sigma_i P_i} = \sigma_i(Z)\big|_{\sigma_i P_i}$ and, because σ_i is bijective, also $\sigma_i(Z)\big|_{\sigma_i P_i} = \sigma_i\big(Z\big|_{P_i}\big)$. Again using that σ_i is bijective implies that $\sigma_i\big(Z\big|_{P_i}\big) \in \sigma_i P_i$ if and only if $Z\big|_{P_i} \in P_i$. This results in the claimed adversary structure

$$\mathcal{Z}^* = \Big\{ Z \subseteq P^* \colon \Big(Z\big|_{P \backslash R} \cup \Big\{ p_{r_i} \in R \colon Z\big|_{P_i} \notin \mathcal{Z}_i \Big\} \Big) \in \mathcal{Z} \Big\} .$$

□

4 Complete Characterization of Tolerable Adversary Structures

4.1 Completeness Theorems

Theorem 2 *In the passive model a set P of processors can compute every function/specification (perfectly) $\mathcal{Z}$-securely if no two sets in the adversary structure $\mathcal{Z}$ cover P (i.e., if $Q^{(2)}(P,\mathcal{Z})$ is satisfied). This bound is tight: if two sets cover P, then there exist functions that cannot be computed $\mathcal{Z}$-securely. The computation is polynomial in the size of the basis $|\overline{\mathcal{Z}}|$ of the adversary structure.*

Proof. We first prove the sufficiency of the condition $Q^{(2)}(P,\mathcal{Z})$ for the existence of $\mathcal{Z}$-secure protocols, and then prove its necessity. The proof that every function can be computed $\mathcal{Z}$-securely if $Q^{(2)}(P,\mathcal{Z})$ is satisfied proceeds in three steps: We describe a construction of a protocol generator, prove the suitability of the construction, and demonstrate its efficiency.

CONSTRUCTION. Consider a set P of processors and a structure $\mathcal{Z}$ for this set P such that $Q^{(2)}(P,\mathcal{Z})$ is satisfied. We construct a $\mathcal{Z}$-secure protocol generator G for the set P of processors, i.e., G takes as input an arbitrary specification (π_0,τ) for the set P_0 of processors and outputs a protocol π for the set $\big(P_0\setminus\{\tau\}\big)\cup P$ of processors that A-securely computes the specification (π_0,τ) for every adversary A with $Z_A|_P\in\mathcal{Z}$.

If some processor $p\in P$ does not occur in any set of $\mathcal{Z}$ (i.e., $\mathcal{Z}|_{\{p\}}=\{\emptyset\}$) then G simply replaces the trusted party τ in the specification by this processor. More precisely, let ρ be the processor mapping that maps τ to p (and is the identity function for all other processors), then $G=((\pi_0,\tau)\mapsto\rho(\pi_0))$.

Consider the case where every processor in P occurs in at least one set in $\mathcal{Z}$. The following construction is based on ideas in [AR63, pp. 22–24] and [Fit96]. We select some three-partition of $\overline{\mathcal{Z}}$ where the size of each set of the partition is at most $\lceil|\overline{\mathcal{Z}}|/3\rceil$. Let $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$ be the union of the first two, the first and the third, and the last two sets of the partition, respectively, each completed such that it is monotone. Assume that protocol generators G_1, G_2, and G_3, each among the set P of processors, tolerating $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$, respectively, have been constructed (by recursion). The protocol generator G that tolerates $\mathcal{Z}$ can be constructed as follows: Remember that G^{p3} is the BGW protocol generator of [BGW88] for the passive model for the set $P_{G^{\mathsf{p3}}}=\{p_1,p_2,p_3\}$ of processors, tolerating the adversary structure $\mathcal{Z}_{G^{\mathsf{p3}}}=\{\{p_1\},\{p_2\},\{p_3\}\}$. Let σ be a bijective processor mapping $\sigma: P_{G^{\mathsf{p3}}}\to\overline{P}$, where $\overline{P}$ is a set of new processor names. First, the protocol generator G applies $\sigma(G^{\mathsf{p3}})$ to the specification (π_0,τ). $\sigma(G^{\mathsf{p3}})$ is a protocol generator that tolerates $\sigma(\mathcal{Z}_{G^{\mathsf{p3}}})$ (Corollary 1), thus the resulting protocol tolerates all adversaries A with $|Z_A|_{\overline{P}}|\leq 1$. Then G simultaneously simulates all three processors in $\overline{P}$ by subprotocols, applying the protocol generators G_1, G_2, and G_3. This results in a protocol π^* for the set P^* of processors that $\mathcal{Z}^*$-securely computes the specification (π_0,τ), where according[18] to Theorem 1 the set of processors is

$$P^*=\Big(\big((P_0\setminus\{\tau\})\cup\overline{P}\big)\setminus\overline{P}\Big)\ \cup\ P=\Big(P_0\setminus\{\tau\}\Big)\cup P$$

[18]Note that in Theorem 1 the protocol generators for the simulation are assumed to be BGW protocol generators. The protocol generators G_1, ... ,G_3 of this proof are recursively constructed protocol generators, which means that in fact only BGW protocol generators (alternated with processor mappings) are applied.

and the tolerated adversary structure is

$$\begin{aligned}\mathcal{Z}^* &= \left\{Z \subseteq P^* \colon \left(Z\big|_{\left((P_0\setminus\{\tau\})\cup\overline{P}\right)\setminus\overline{P}} \cup \{p_{r_i} \in \overline{P} \colon Z|_P \notin \mathcal{Z}_i\}\right) \in \sigma\mathcal{Z}_{G^{\mathsf{p3}}}\right\} \\ &= \left\{Z \subseteq P^* \colon \left|\left(Z\big|_{P_0\setminus\{\tau\}} \cup \{p_{r_i} \in \overline{P} \colon Z|_P \notin \mathcal{Z}_i\}\right)\Big|_{\overline{P}}\right| \leq 1\right\} \\ &= \left\{Z \subseteq P^* \colon \left|\{p_{r_i} \in \overline{P} \colon Z|_P \notin \mathcal{Z}_i\}\right| \leq 1\right\}.\end{aligned}$$

Every set $Z \in \mathcal{Z}$ is in two of the structures $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$, thus every adversary A with $Z_A|_P \in \mathcal{Z}$ is tolerated in π^*. As claimed, the constructed protocol generator G is for the set P of processors and is $\mathcal{Z}$-secure.

SUITABILITY. The suitability of this construction can be proved by induction. First, consider an adversary structure $\mathcal{Z}$ satisfying $Q^{(2)}$ with $|\overline{\mathcal{Z}}| \leq 2$. Since the (at most) two sets in $\overline{\mathcal{Z}}$ do not cover P, and all other sets in $\mathcal{Z}$ are subsets of one of the sets in the basis, there is a processor $p \in P$ that does not occur in any set in $\mathcal{Z}$ (induction basis). Now assume that we can construct a protocol generator for every adversary structure which contains $2m$ of the sets in $\overline{\mathcal{Z}}$ (induction hypothesis). Then the above construction yields a protocol generator for an arbitrary adversary structure with up to $3m$ of the sets in $\overline{\mathcal{Z}}$ (induction step).

Let t_i be defined as the basis size guaranteed to be achievable with recursion of depth i. We have $t_0 = 2$, $t_1 = 3$, and $t_{i+1} = t_i + \lfloor t_i/2 \rfloor$. One can easily verify that $(3/2)^i \leq t_i \leq (3/2)^{i+2}$. Thus, in order to construct a protocol that tolerates the adversary structure $\mathcal{Z}$, the recursion depth is at most $\lceil \log_{3/2} |\overline{\mathcal{Z}}| \rceil$.

EFFICIENCY. The protocol generator G^{p3} applied to a specification (π, p) translates every statement in π that involves p into a statement sequence of length at most b, where b is a constant parameter of G^{p3}. Considering all simultaneous simulations at a given level i of the recursion, every statement is affected by the application of at most two BGW protocol generators (because every statement involves at most two processors). Hence the total blow-up due to a given level of the recursion is at most b^2. The total length of the constructed protocol tolerating $\mathcal{Z}$ is thus at most $|\pi_0| \cdot \left(b^2\right)^{\lceil \log_{3/2} |\overline{\mathcal{Z}}| \rceil} = |\pi_0| \cdot |\overline{\mathcal{Z}}|^{O(1)}$, which is polynomial in $|\overline{\mathcal{Z}}|$.

In order to prove the necessity of the condition $Q^{(2)}(P, \mathcal{Z})$ for the existence of $\mathcal{Z}$-secure protocols, suppose there is a protocol that tolerates an adversary structure not satisfying $Q^{(2)}$, i.e., there are two potential sets Z_1 and Z_2 with $Z_1 \cup Z_2 = P$. Without loss of generality we assume $Z_1 \cap Z_2 = \emptyset$. Then we can construct a protocol with two processors p_1 and p_2, where p_1 simulates all processors in Z_1 and p_2 simulates all processors in Z_2 (i.e., we apply a mapping to the given protocol), and we obtain a protocol for two processors that tolerates both sets with a single adverse processor. Such a protocol for secure function evaluation does not exist for most functions (for example for the binary OR-function), as stated in [BGW88], thus resulting in a contradiction. A more careful analysis of the class of functions that are not securely computable if $Q^{(2)}$ is not satisfied is given in [CK89], [Kus89], and [Bea89]. □

Theorem 3 *In the active model a set P of processors can compute every function/specification (perfectly) $\mathcal{Z}$-securely if no three sets in the adversary structure $\mathcal{Z}$ cover P (i.e., if $Q^{(3)}(P, \mathcal{Z})$ is satisfied). This bound is tight: if three sets cover the full set of processors, there are functions that cannot be computed $\mathcal{Z}$-securely. The computation is polynomial in the size of the basis $|\overline{\mathcal{Z}}|$ of the adversary structure.*

Proof (Sketch). This proof for sufficiency of $Q^{(3)}(P, \mathcal{Z})$ is along the lines of the proof of Theorem 2 and also proceeds in the same three steps. We describe only the major differences.
CONSTRUCTION. A four-partition of the adversary structure $\overline{\mathcal{Z}}$ is selected where the size of each set of the partition is at most $\lceil |\overline{\mathcal{Z}}|/4 \rceil$. By recursion, a protocol is constructed for each of the four unions of three set of the partition. First, the protocol generator applies G^{a4} in order to substitute the trusted party τ in the specification by a protocol among four virtual processors, then simultaneously replaces the four virtual processors by applying the recursively constructed protocol generators. Applying Theorem 1 shows that the tolerated adversary structure is $\mathcal{Z}$.
SUITABILITY. The induction basis (there is a processor $p \in P$ that does not occur in $\mathcal{Z}$) holds for any structure $\mathcal{Z}$ with $|\overline{\mathcal{Z}}| \leq 3$, and the induction step constructs a protocol generator that tolerates $4m$ of the sets in $\mathcal{Z}$ by assuming protocol generators that tolerate $3m$ of the sets.
EFFICIENCY. Let b be the constant "blow-up factor" of G^{a4}, and let u_i be defined as the minimal size of the basis of the adversary structures guaranteed to be achievable with recursion of depth i. The sequence u_i is hence given by $u_0 = 3$, $u_1 = 4$, and $u_{i+1} = u_i + \lfloor u_i/3 \rfloor$. One can easily verify that $(4/3)^i \leq u_i \leq (4/3)^{i+3}$. Thus, in order to construct a protocol that tolerates the adversary structure $\mathcal{Z}$, the recursion depth is at most $\lceil \log_{4/3} |\overline{\mathcal{Z}}| \rceil$, and the total length of the constructed protocol tolerating $\mathcal{Z}$ is at most $|\pi_0| \cdot \left(b^2\right)^{\lceil \log_{4/3} |\overline{\mathcal{Z}}| \rceil} = |\pi_0| \cdot |\overline{\mathcal{Z}}|^{O(1)}$, which is polynomial in $|\overline{\mathcal{Z}}|$.

In order to prove the necessity of condition $Q^{(3)}(P, \mathcal{Z})$, suppose that there exists a protocol generator for an adversary structure not satisfying $Q^{(3)}$, i.e., there are three potential adversaries that cover the full set of processors. Then we can construct a protocol among three processors, where each of them simulates the processors of one adversary, and we obtain a protocol among three processors, perfectly tolerating active cheating of one of them. Such a protocol for secure function evaluation does not exist for most functions (for example for the broadcast function, as proved in [PSL80] and [LSP82]), thus resulting in a contradiction. $\square$

4.2 Example

We apply Theorem 2 to construct a protocol generator G for the passive model among the set $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ of processors that tolerates the adversary structure $\mathcal{Z}$ with the basis $\overline{\mathcal{Z}} = \{\{p_1, p_4, p_6\}$, $\{p_2, p_3, p_6\}$, $\{p_1, p_2, p_6\}$, $\{p_1, p_2, p_5\}$, $\{p_2, p_4, p_5\}$ $\{p_1, p_3, p_5\}$, $\{p_1, p_2, p_3, p_4\}\}$. It is easy to verify that $Q^{(2)}(P, \mathcal{Z})$ is satisfied.

As a short notation, we write $[p_i, p_j, p_k]$ for the (mapped) protocol generator G^{p3} with the three processors p_i, p_j, and p_k, and $[p_i, p_j, [p_k, p_l, p_m]]$ for the protocol generator among the processors p_i, p_j and a virtual processor simulated by a protocol generated by the protocol generator G^{p3} among the processors p_k, p_l, and p_m (i.e., a mapped protocol generator). As a special case, $[p]$ refers to the protocol generator that simply replaces the name of the trusted party in the multiparty computation specification by p. Whenever a structure is partitioned, this partition is not made explicit, but can easily be derived from the three resulting structures.

Step 1: Divide $\overline{\mathcal{Z}}$ into three partitions and set
$\overline{\mathcal{Z}}_1 = \{\{p_1, p_4, p_6\}, \{p_2, p_3, p_6\}, \{p_1, p_2, p_6\}, \{p_1, p_2, p_5\}, \{p_2, p_4, p_5\}\}$,
$\overline{\mathcal{Z}}_2 = \{\{p_1, p_4, p_6\}, \{p_2, p_3, p_6\}, \{p_1, p_2, p_6\}, \{p_1, p_3, p_5\}, \{p_1, p_2, p_3, p_4\}\}$,
$\overline{\mathcal{Z}}_3 = \{\{p_1, p_2, p_5\}, \{p_2, p_4, p_5\}, \{p_1, p_3, p_5\}, \{p_1, p_2, p_3, p_4\}\}$.

Step 2: Construct G_1 tolerating $\overline{\mathcal{Z}}_1$.

Step 2.1: Divide $\overline{\mathcal{Z}}_1$ into three partitions and set
$\overline{\mathcal{Z}}_{11} = \{\{p_1, p_4, p_6\}, \{p_2, p_3, p_6\}, \{p_1, p_2, p_6\}\}$,
$\overline{\mathcal{Z}}_{12} = \{\{p_1, p_4, p_6\}, \{p_1, p_2, p_5\}, \{p_2, p_4, p_5\}\}$,
$\overline{\mathcal{Z}}_{13} = \{\{p_2, p_3, p_6\}, \{p_1, p_2, p_6\}, \{p_1, p_2, p_5\}, \{p_2, p_4, p_5\}\}$.
Step 2.2: Construct G_{11} tolerating $\overline{\mathcal{Z}}_{11}$. This is achieved by $[p_5]$.
Step 2.3: Construct G_{12} tolerating $\overline{\mathcal{Z}}_{12}$. This is achieved by $[p_3]$.
Step 2.4: Construct G_{13} tolerating $\overline{\mathcal{Z}}_{13}$.

Step 2.4.1: Divide $\overline{\mathcal{Z}}_{13}$ into three partitions and set
$\overline{\mathcal{Z}}_{131} = \{\{p_2, p_3, p_6\}, \{p_1, p_2, p_6\}, \{p_1, p_2, p_5\}\}$,
$\overline{\mathcal{Z}}_{132} = \{\{p_2, p_3, p_6\}, \{p_2, p_4, p_5\}\}$,
$\overline{\mathcal{Z}}_{133} = \{\{p_1, p_2, p_6\}, \{p_1, p_2, p_5\}, \{p_2, p_4, p_5\}\}$.
Step 2.4.2: Construct G_{131} tolerating $\overline{\mathcal{Z}}_{131}$. This is achieved by $[p_4]$.
Step 2.4.3: Construct G_{132} tolerating $\overline{\mathcal{Z}}_{132}$. This is achieved by $[p_1]$.
Step 2.4.4: Construct G_{133} tolerating $\overline{\mathcal{Z}}_{133}$. This is achieved by $[p_3]$.
Step 2.4.5: $G_{13} = [p_4, p_1, p_3]$ is $\overline{\mathcal{Z}}_{13}$-secure.

Step 2.5: $G_1 = [p_5, p_3, [p_4, p_1, p_3]]$ is $\overline{\mathcal{Z}}_1$-secure.

Step 3: Construct G_2 tolerating $\overline{\mathcal{Z}}_2$.

Step 3.1: Divide $\overline{\mathcal{Z}}_2$ into three partitions and set
$\overline{\mathcal{Z}}_{21} = \{\{p_1, p_4, p_6\}, \{p_2, p_3, p_6\}, \{p_1, p_2, p_6\}\}$,
$\overline{\mathcal{Z}}_{22} = \{\{p_1, p_4, p_6\}, \{p_1, p_3, p_5\}, \{p_1, p_2, p_3, p_4\}\}$,
$\overline{\mathcal{Z}}_{23} = \{\{p_2, p_3, p_6\}\}, \{p_1, p_2, p_6\}, \{p_1, p_3, p_5\}, \{p_1, p_2, p_3, p_4\}\}$.
Step 3.2: Construct G_{21} tolerating $\overline{\mathcal{Z}}_{21}$. This is achieved by $[p_5]$.
Step 3.3: Construct G_{22} tolerating $\overline{\mathcal{Z}}_{22}$.

Step 3.3.1: Divide $\overline{\mathcal{Z}}_{22}$ into three partitions and set
$\overline{\mathcal{Z}}_{221} = \{\{p_1, p_4, p_6\}, \{p_1, p_3, p_5\}\}$,
$\overline{\mathcal{Z}}_{222} = \{\{p_1, p_4, p_6\}, \{p_1, p_2, p_3, p_4\}\}$,
$\overline{\mathcal{Z}}_{223} = \{\{p_1, p_3, p_5\}, \{p_1, p_2, p_3, p_4\}\}$.
Step 3.3.2: Construct G_{221} tolerating $\overline{\mathcal{Z}}_{221}$. This is achieved by $[p_2]$.
Step 3.3.3: Construct G_{222} tolerating $\overline{\mathcal{Z}}_{222}$. This is achieved by $[p_5]$.
Step 3.3.4: Construct G_{223} tolerating $\overline{\mathcal{Z}}_{223}$. This is achieved by $[p_6]$.
Step 3.3.5: $G_{22} = [p_2, p_5, p_6]$ is $\overline{\mathcal{Z}}_{22}$-secure.

Step 3.4 Construct G_{23} tolerating $\overline{\mathcal{Z}}_{23}$.

Step 3.4.1: Divide $\overline{\mathcal{Z}}_{23}$ into three partitions and set
$\overline{\mathcal{Z}}_{231} = \{\{p_2, p_3, p_6\}, \{p_1, p_2, p_6\}, \{p_1, p_3, p_5\}\}$,
$\overline{\mathcal{Z}}_{232} = \{\{p_2, p_3, p_6\}, \{p_1, p_2, p_6\}, \{p_1, p_2, p_3, p_4\}\}$,
$\overline{\mathcal{Z}}_{233} = \{\{p_1, p_3, p_5\}, \{p_1, p_2, p_3, p_4\}\}$.
Step 3.4.2: Construct G_{231} tolerating $\overline{\mathcal{Z}}_{231}$. This is achieved by $[p_4]$.
Step 3.4.3: Construct G_{232} tolerating $\overline{\mathcal{Z}}_{232}$. This is achieved by $[p_5]$.
Step 3.4.4: Construct G_{233} tolerating $\overline{\mathcal{Z}}_{233}$. This is achieved by $[p_6]$.
Step 3.4.5: $G_{23} = [p_4, p_5, p_6]$ is $\overline{\mathcal{Z}}_{23}$-secure.

Step 3.5: $G_2 = [p_5, [p_2, p_5, p_6], [p_4, p_5, p_6]]$ is $\overline{\mathcal{Z}}_2$-secure.

Step 4: Construct G_3 tolerating $\overline{\mathcal{Z}}_3$. This is achieved by $[p_6]$.

Step 5: The protocol generator

$$G = \Big[[p_5, p_3, [p_4, p_1, p_3]], [p_5, [p_2, p_5, p_6], [p_4, p_5, p_6]], p_6 \Big]$$

is $\mathcal{Z}$-secure.

Figure 3 illustrates this protocol generator. Remember that p_i for $i < 0$ refers to a virtual processor and does not explicitly appear in the above description.

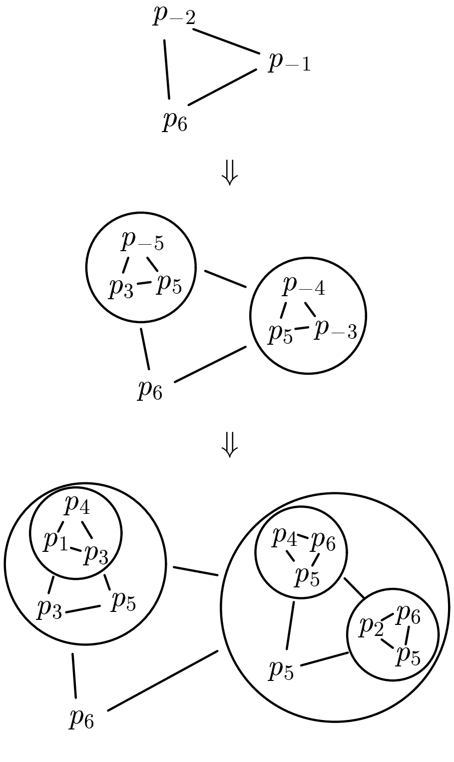

Figure 3: An example of recursive processor simulation.

5 Adversary Structures without Efficient Protocols

The goal of this section is, informally, to prove that there exists a family of adversary structures for which the length of every resilient protocol grows exponentially in the number of processors.

For a specification (π_0, τ), a set P of processors, and an adversary structure $\mathcal{Z}$, let $\varphi\big((\pi_0, \tau), P, \mathcal{Z}\big)$ denote the length of the shortest protocol π for P that $\mathcal{Z}$-securely computes (π_0, τ). Furthermore, let (π_*, τ) denote the specification for the processors p_1 and p_2 that reads one input of both processors, computes the product, and hands it to p_1. Finally, let P_n denote the set $\{p_1, \ldots, p_n\}$ of processors.

Theorem 4 *For both the passive and the active model there exists a family $\mathcal{Z}_2, \mathcal{Z}_3, \ldots$ of adversary structures for the sets $P_2, P_3, \ldots$ of processors, respectively, such that the length $\varphi\big((\pi_*, \tau), P_n, \mathcal{Z}_n\big)$ of the shortest $\mathcal{Z}_n$-secure protocol for (π_*, τ) grows exponentially in n.*

In order to prove the theorem we need an additional definition: An admissible adversary structure $\mathcal{Z}$ for the set P of processors is *maximal* if $Q^{(2)}(P, \mathcal{Z})$ (in the passive model) or $Q^{(3)}(P, \mathcal{Z})$ (in the active model) is satisfied, but any adversary structure $\mathcal{Z}'$ with $\mathcal{Z} \subset \mathcal{Z}'$ (and $\mathcal{Z} \neq \mathcal{Z}'$) violates $Q^{(2)}(P, \mathcal{Z}')$, or $Q^{(3)}(P, \mathcal{Z}')$, respectively.

Proof. The proof proceeds in three steps: First we prove that in both models, the number of maximal admissible adversary structures grows doubly exponentially in the number n of processors. In the second step we show that for the given specification (π_*, τ), for every maximal admissible adversary structure a different protocol is required. Finally, we conclude that for some adversary structures the length of every secure protocol is exponential in the number of processors.

1. First consider the passive model. Without loss of generality, assume that $n = |P|$ is odd, and let $m = (n+1)/2$. Fix a processor $p \in P$, and consider the set $\mathcal{B}$ that contains all subsets of $P \backslash \{p\}$ with exactly m processors, i.e., $\mathcal{B} = \{Z \subseteq (P \backslash \{p\}) : |Z| = m\}$. For each subset $\mathcal{B}' \subseteq \mathcal{B}$, we define $\mathcal{Z}_{\mathcal{B}'}$ to be the adversary structure that contains all sets in $\mathcal{B}'$, plus all sets $Z \subseteq P$ with $|Z| < n/2$ and $(P \backslash Z) \notin \mathcal{B}$. One can easily verify that $\mathcal{Z}_{\mathcal{B}'}$ is admissible and maximal, and that for two different subsets $\mathcal{B}', \mathcal{B}'' \subseteq \mathcal{B}$, the structures $\mathcal{Z}_{\mathcal{B}'}$ and $\mathcal{Z}_{\mathcal{B}''}$ are different. The size of $\mathcal{B}$ is $|\mathcal{B}| = \binom{n-1}{m} = 2^{\Omega(n)}$, hence there are $2^{2^{\Omega(n)}}$ different subsets $\mathcal{B}'$ of $\mathcal{B}$, and thus doubly exponentially many different maximal admissible adversary structures for the passive model.

   For the active model, consider an arbitrary maximal admissible adversary structure $\mathcal{Z}$ for P for the passive model, i.e., $Q^{(2)}(P, \mathcal{Z})$ is satisfied. Clearly, for an additional processor $p \notin P$, the structure $\mathcal{Z} \cup \{\{p\}\}$ for the set $P \cup \{p\}$ is admissible for the active model (i.e., $Q^{(3)}(P \cup \{p\}, \mathcal{Z} \cup \{\{p\}\})$), and there exists a maximal adversary structure $\widehat{\mathcal{Z}} \supseteq (\mathcal{Z} \cup \{\{p\}\})$ for $P \cup \{p\}$. For two different adversary structures $\mathcal{Z}$ and $\mathcal{Z}'$, also $\widehat{\mathcal{Z}}$ and $\widehat{\mathcal{Z}'}$ are different (one can easily compute $\mathcal{Z}$ for a given $\widehat{\mathcal{Z}}$). Hence, the number of maximal admissible adversary structures for the active model with n processors is at least as large as for the passive model with $n-1$ processors, thus doubly exponential in n.

2. Let $\mathcal{Z}$ be a maximal admissible adversary structure, and let π be a protocol that $\mathcal{Z}$-securely computes (π_*, τ). For the sake of contradiction, assume that for some other maximal admissible adversary structure $\mathcal{Z}'$ (where $\mathcal{Z}' \neq \mathcal{Z}$), the same protocol π $\mathcal{Z}'$-securely computes (π_*, τ). Then π would $(\mathcal{Z} \cup \mathcal{Z}')$-securely compute (π_*, τ). However, since both $\mathcal{Z}$ and $\mathcal{Z}'$ are maximal admissible, $(\mathcal{Z} \cup \mathcal{Z}')$ is not admissible, and hence no such protocol exists (see Theorems 2 and 3). Hence, for each maximal admissible adversary structure $\mathcal{Z}$ a different protocol π is required for securely computing (π_*, τ).

3. There are doubly exponentially many maximal admissible adversary structures, and for each of them, a different protocol is required, hence there are doubly exponentially many different protocols. This implies that some of these protocols have exponential length. $\square$

6 Conclusions and Open Problems

We have given a complete characterization of adversaries tolerable in unconditional multiparty computation. Corresponding results for the case of cryptographic security are given in [CDM98] where also an alternative proof technique for the unconditional case based on span-program secret-sharing schemes is presented. Our techniques also allow us to prove the natural generalization of the threshold-type results in [RB89] for a model with a broadcast channel: unconditional multiparty computation is possible if and only if no two sets in the adversary structure cover the full player set [HM97], [SS98]. More generally, the simulation technique applies to most previously proposed unconditional multiparty protocols. Furthermore, we believe that every reasonable protocol generator can be used in our construction, but we have also given an example of an artificial protocol generator which can not (see footnote 16). Formulating the exact condition for when a protocol generator can be applied in our construction is suggested as an open problem. The player substitution techniques can also be applied in the cryptographic model of multiparty computation, but the security of such composite protocols remains to be proven [Can98b], [Bea91a], [MR91].

The efficiency of the proposed protocols is polynomial in the size of the basis of the adversary structure to be tolerated. It is an open problem to find other general descriptions of structures for which polynomial (in the number of players) protocols can be found (for a possible approach and some new results see [CDM98]). A further open problem is to give general conditions on adversary structures such that polynomial protocols exist. However, the number of maximal bases of structures satisfying the $Q^{(2)}$ or the $Q^{(3)}$ condition are more than exponential in the number of processors, and therefore a construction of polynomial protocols can be found at most for some particular classes of structures.

The recursive construction of Section 4 has a large number of degrees of freedom in the partitioning of the adversary structure. We did not investigated the problem of finding recursive partitionings with high or optimal efficiency, nor the round complexity of our protocols.

Acknowledgments

We would like to thank Matthias Fitzi for his collaboration on finding efficient protocols and for many helpful discussions. We are very grateful to Oded Goldreich for his detailed comments on the conference version of this paper [HM97] and for insisting on a sufficiently formal treatment, and to the anonymous referees for their constructive comments. Ran Canetti provided an early unpublished version of [Can98b], where a definition of security of unconditional multiparty protocols is given. We thank Masayuki Abe, Christian Cachin, Ronald Cramer, Claude Crépeau, Ivan Damgård, Rosario Gennaro, Tal Rabin, Markus Stadler, and Stefan Wolf for interesting discussions related to this paper.

References

[AR63] S. Akers and T. Robbins. Logical design with three-input majority gates. *Computer Design*, pp. 12–27, Mar. 1963.

[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In *Proc. 8th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 201–210, Aug. 1989.

[BDDS92] A. Bar-Noy, D. Dolev, C. Dwork, and H. R. Strong. Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement. *Information and Computation*, 97(2):205–233, Apr. 1992.

[Bea89] D. Beaver. Perfect privacy for two-party protocols. In *Proc. DIMACS Workshop on Distributed Computing and Cryptography*, Oct. 1989.

[Bea91a] D. Beaver. Foundations of secure interactive computing. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pp. 377–391. Springer-Verlag, Berlin, 1991.

[Bea91b] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2): 75–122, 1991.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pp. 1–10, 1988.

[BL88] J. C. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pp. 27–35. Springer-Verlag, Berlin 1988.

[BW98] D. Beaver and A. Wool. Quorum-based secure multi-party computation. In *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pp. 346–360. Springer-Verlag, Berlin, 1998.

[Can95] R. Canetti. *Studies in Secure Multiparty Computation and Applications*. Ph.D. thesis, Weizmann Institute of Science, Rehovot 76100, June 1995.

[Can98a] R. Canetti. Personal communications, 1998.

[Can98b] R. Canetti. Security and composition of multi-party cryptographic protocols. Manuscript, June 1998. Former (more general) version: Modular composition of multi-party cryptographic protocols, Nov. 1997.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pp. 11–19, 1988.

[CDG87] D. Chaum, I. Damgård, and J. van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pp. 87–119. Springer-Verlag, Berlin, 1987.

[CDM98] R. Cramer, I. Damgård, and U. Maurer. Span programs and general multi-party computation. Manuscript, 1998.

[CFGN96] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *Proc. 28th ACM Symposium on the Theory of Computing (STOC)*, pp. 639–648, Nov. 1996.

[CFSY96] R. Cramer, M. K. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pp. 72–83. IACR/Springer-Verlag, Berlin, May 1996.

[CG96] R. Canetti and R. Gennaro. Incoercible multiparty computation. In *Proc. 37th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 504–513, 1996.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proc. 36th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 41–51, Oct. 1995.

[CGT95] C. Crépeau, J. van de Graaf, and A. Tapp. Committed oblivious transfer and private multiparty computation. In *Advances in Cryptology — CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pp. 110–123. Springer-Verlag, Berlin, 1995.

[CH94] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pp. 425–438. Springer-Verlag, Berlin, 1994.

[Cha89] D. Chaum. The spymasters double-agent problem. In *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pp. 591–602. Springer-Verlag, Berlin, 1989.

[CK89] B. Chor and E. Kushilevitz. A zero-one law for Boolean privacy. In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pp. 62–72, 1989.

[CKOR97] R. Canetti, E. Kushilevitz, R. Ostrovsky, and A. Rosén. Randomness vs. fault-tolerance. In *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 35–44, Aug. 1997.

[dDFY94] A. de Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *Proc. 26th ACM Symposium on the Theory of Computing (STOC)*, pp. 522–533, 1994.

[FHM98] M. Fitzi, M. Hirt, and U. Maurer. Trading correctness for privacy in unconditional multiparty computation. In *Advances in Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, 1998.

[FHM99] M. Fitzi, M. Hirt, and U. Maurer. General adversaries in unconditional multi-party computation. Manuscript, 1999.

[Fit96] M. Fitzi. Erweiterte Zugriffstrukturen in Multi-Party-Computation. Student's project, 1996.

[FKN94] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation. In *Proc. 26th ACM Symposium on the Theory of Computing (STOC)*, pp. 554–563, 1994.

[FM88] P. Feldman and S. Micali. Optimal algorithms for Byzantine agreement. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pp. 148–161, 1988.

[FM97] P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, Aug. 1997.

[FM98] M. Fitzi and U. Maurer. Efficient Byzantine agreement secure against general adversaries. In *Distributed Computing — DISC '98*, volume 1499 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Sept. 1998.

[FR96] M. K. Franklin and M. K. Reiter. The design and implementation of a secure auction service. *IEEE Transactions on Software Engineering*, 22(5):302–312, May 1996.

[FY92] M. K. Franklin and M. Yung. Communication complexity of secure computation. In *Proc. 24th ACM Symposium on the Theory of Computing (STOC)*, pp. 699–710, 1992.

[Gen96] R. Gennaro. *Theory and Practice of Verifiable Secret Sharing*. Ph.D. thesis, Massachusetts Institute of Technology (MIT), Cambridge, MA, May 1996.

[GHY87] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: secure fault-tolerant protocols and the public-key model. In *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pp. 135–155. Springer-Verlag, Berlin, 1987.

[GJKR96a] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pp. 157–172. Springer-Verlag, Berlin, Aug. 1996.

[GJKR96b] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In U. Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pp. 354–371. Springer-Verlag, Berlin, May 1996.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, pp. 218–229, 1987.

[GRR98] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC)*, 1998.

[HM97] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multiparty computation. In *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 25–34, Aug. 1997.

[ISN87] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *Proceedings IEEE Globecom '87*, pp. 99–102. IEEE, New-York, 1987.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single data base computationally-private information retrieval. In *Proc. 38th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 364–373, 1997.

[Kus89] E. Kushilevitz. Privacy and communication complexity (extended abstract). In *Proc. 30th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 416–421, 1989.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[MR91] S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pp. 392–404. Springer-Verlag, Berlin, 1991.

[OY91] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 51–59, Aug. 1991.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, Apr. 1980.

[Rab94] T. Rabin. Robust sharing of secrets when the dealer is honest or cheating. *Journal of the ACM*, 41(6):1089–1109, Nov. 1994.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pp. 73–85, 1989.

[SS98] A. Smith and A. Stiglic. Multiparty computation unconditionally secure against Q^2 adversary structures. Manuscript, July 1998.

[Yao82] A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 160–164. IEEE, New-York, 1982.