HILDESHEIMER
INFORMATIK-
BERICHTE

ISSN 0941-3014

Birgit Pfitzmann
Michael Waidner

A General Framework for Formal Notions of "Secure" Systems

11/94 (April 1994)

UNIVERSITÄT
HILDESHEIM

Dieser Bericht ist herausgegeben vom

Institut für Informatik

Postfach 10 13 63
31113 Hildesheim

A General Framework for Formal Notions of "Secure" Systems

Birgit Pfitzmann

Universität Hildesheim
Institut für Informatik
Marienburger Platz 22
D-31141 Hildesheim, Germany

pfitzb@informatik.uni-hildesheim.de

Michael Waidner

Universität Karlsruhe
Fakultät für Informatik
Zirkel 2
D-76128 Karlsruhe, Germany

waidner@ira.uka.de

Abstract

Research about secure systems takes place in several largely unrelated communities without much mutual understanding, and without a common notion of "security". Example areas are access control and information flow control, cryptology, secure software, statistical databases, and fault tolerance and safety. We strive to unify these approaches, or at least to give them a common understanding of their relations, by presenting a general framework for the design of secure systems and showing how the main research activities of several example areas fit together in it.

Our main issue is not terminology, but the notions of security as such, and how such security can be achieved. We keep formalism to a minimum and use simple terminology from general computer science, avoiding all the jargons of the communities involved. Nevertheless, in contrast to previous approaches, our goal was to identify notions that can be defined formally (at least for those design stages that are accessible to formalization at all), and to relate them to usual formal notions about system design, e.g., "specification" and "correctness".

Particular notions we discuss are trust models, the requirement that a system does "nothing else" than what is specified, and the interplay of individual goals and a global specification. Our approach is not restricted to systems with a trusted central authority or computing base.

Keywords

Theoretical Foundations of Security, Specification, Security Semantics, Cryptology

Contents

Figures

1 Introduction

There are different informal definitions of security. However, most people might agree that a secure IT-system (information technology) should be correct, but not only that. This is enough of a starting point.[1] We show a design framework where one can see what aspects beyond usual correctness can be considered, how they are related, and which research areas actually treat them.

Our intentions are threefold:

- To promote understanding between different communities dealing with security.
- To give some notions a more rigorous meaning and to discuss their limitations, e.g., trust models and the notion that a system does nothing else than what is specified.
- To emphasize notions that are missing in current security evaluation criteria such as [ITSEC2_91] (at least in their "example functionality classes" and "generic headings" for security enforcing functions). For instance, these criteria only treat systems with a trusted central authority or computing base, and largely concentrate on access control models for their specifications.

We think that no previous attempt to identify notions of security that are both general and formalizable has been made. Formal, but specific ones are mentioned later. Recent general, but informal research is listed here: Security in a very broad sense was investigated by the ESPRIT Project PDCS (Predictably Dependable Computing Systems) [RaLa_92], but as far as we know, PDCS has not created formal definitions outside reliability and safety. Very general papers about security (but no so much reliability and safety) are [Dier2_91, Bisk_93]. We present some more concepts than those and claim that our concepts are much nearer to formalization. Most other papers about the notion of "security" are limited to the classical aspects of access and information flow control, as outlined, e.g., in [Land_81, GoMe_82], or at least to centralized systems [Jaco_88]. In particular, this is true for current security evaluation criteria, such as [ITSEC2_91]. Similarly, results of the German REMO project, which has a rather broad informal scope, are restricted to either the verification of sequential programs or to access control as soon as they are made more precise, e.g., [Mund_93]. None of these models covers all ongoing research, in particular not that in cryptology, e.g., secure payment systems [Cha8_85, PWP_90], generalized digital signature schemes [Pfit4_93], and secure distributed function evaluation [Yao_82, Beav5_91].

We admit that we started this research because cryptology is our own primary background, but also since we noticed that results from other approaches are usually disregarded in our own field. For instance, notions of security in cryptology are so specific that with new system classes, the new security definition is almost as likely to contain flaws as the proposed realization (see, e.g., [PfWa2_92]).[2]

---

[1] Safety specialists might object that their systems need not be "correct", only fulfil some important properties correctly. We will see that as correctness with respect to a special specification.

[2] The popular logics for the design of cryptologic protocols, following [BuAN_90], are no help for this purpose, since they only start when all cryptology has already been replaced by idealizations (which can be seen as another form of specifications, except that they — based on [DoYa_83] — make more assumptions about cryptologic primitives than those fulfil by their real definitions). Other problems are mentioned, e.g., in [Syve1_93, Pfit4_93].

In Section 2, we give a brief overview of our framework. In Section 3, we show how those areas of security where most research is going on fit into this framework, and thus how they can work together. In Section 4, we sketch formalizations our general notions of security and refer to existing definitions in specific research areas that have carried out parts of this task already.

2 Overview of the Framework By Design Stages

This section surveys the design process of a secure system. Some problems and solutions are simplified; more details are given in Section 4. We strive for completeness with the design stages; in many concrete cases, some stages may be missing. An overview is given in Figure 1.

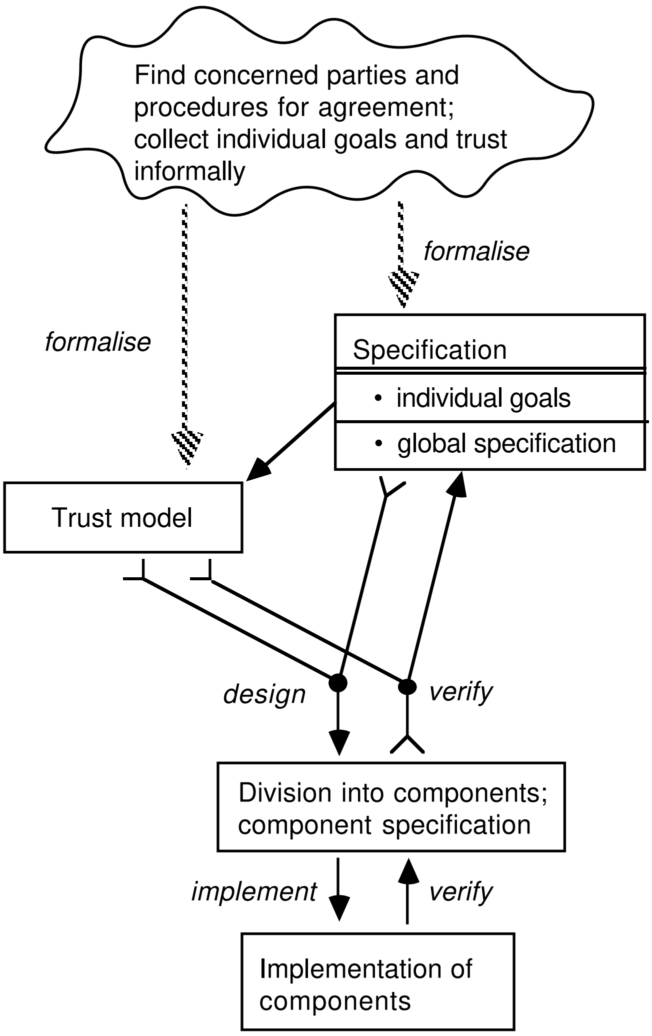

Figure 1 Stages in the design of a secure system.
The terminology is explained in Sections 2.1 to 2.4. Rectangles and normal arrows denote objects and actions of computer science (ideally formal); clouds and dotted arrows are necessarily informal.

2.1 Before a Specification

At the beginning of the design of a secure IT-system, the parties concerned must somehow agree

- who the parties concerned are,
- what each of them desires,
- and how they will try to agree on a common goal.

These questions are largely outside computer science. However, IT-security only makes sense if this agreement really covers the interests of everybody concerned. Unfortunately, this step is missing in current IT-security criteria — just the designer and the future owner of the system are considered — however, e.g., a power plant control or a payment system clearly concern other people, too. Its importance was stressed in [Bisk_93], where "values" and "goals" were distinguished: Values are independent of the system (such as the health of the people concerned) and goals are what the specific system should achieve. This distinction is helpful intuitively, but we do not consider it further in the following sections since we do not see how to formalize it (which was not intended in [Bisk_93] anyway, of course).

Below, we show that considering concerned parties separately is important not only socially, but also in the security specifications. We use the term **concerned party** for everyone who has any wishes about the system, whereas **user** means someone who interacts with the system at its interface; **system** means an IT-system and comprises neither of these people.

Variant for "Products"

So far, it was assumed that a system was designed for a particular application. If one designs components or subsystems for systems that are still unknown (this is called **products** in [ITSEC2_91]), the concerned parties are also unknown. Hence one has to identify **virtual concerned parties** (formally, they are just variables); parties such as owners of components mentioned later are treated similarly.[3] When a product is used in a system, the virtual parties in its specification have to be substituted by real ones, and the real ones have to be asked to agree.

2.2 Specification

Computer science starts, at the latest, when specifications are written. By specification we mean a description of the system behaviour at its external interface, as independent as possible of the internal structure that will be designed. We will not dwell on usual aspects, e.g., that there are specifications with different degrees of formality and that they are developed in cycles. However, there are general security-specific aspects.

Two-Phase Structure

Most importantly, the specification technique has to reflect that there may be several concerned parties, each with their own interests. Thus, in the general case, there is a two-phase structure in the specification:

---

[3] Sometimes the term "role" is used for such virtual parties, but it is also used for other purposes, e.g., for a class of parties with presumably similar goals.

1. There are **individual goals** of individual parties.
2. These individual goals are unified and refined into a **global specification**.

The individual goals need not be in a bijective relation with the parties concerned — a party can have many goals that it cannot immediately specify coherently.

Furthermore, *sets* of concerned parties often have more individual goals than each of them alone. For instance, with a communication system, only sets of at least two users have the goal of being able to communicate.

The global specification should definitely be free of contradictions; if the goals turn out to be contradictory, they have to be modified by discussion between the concerned parties.[4] Furthermore, a global specification is often assumed to be "complete" [Dier2_91, Bisk_93] (where this notion is left to the intuitive understanding of the reader — necessarily, since the notion of specification is so informal that there are several options; see Section 4.3 for our formalization). One reason to require completeness is that this makes an informal review from a different perspective possible, at least if the global specification is in its own specification language, and not just defined as a "completion" of the goals collected. Another reason is the nothing-else aspect discussed below.

Note that we assume that the individual goals are formalized (at least in theory), whereas in [Dier2_91, Bisk_93], the formal process starts with a complete global specification. Besides the fact that a complete specification is a complicated object to produce in one step, we found that the relation between individual goals and global specifications is an important aspect of several research areas within security, and we will show that individual goals are important in combination with trust models.

"Nothing-Else" Aspect

It is sometimes required that a secure system does nothing else than what is specified [Dier2_91, Bisk_93]; this notion, just like completeness, was left vague because of the vague notion of specification. (Our Sections 4.3 and 4.5 mention problems with it and possible formalizations.) This nothing-else aspect is then seen as the main difference to correctness, i.e., the mere notion of fulfilling a specification. In fact, it is important in some applications, e.g., if one requires the absence of Trojan horses or covert channels, but we do not endorse that it is always needed.

Preformalized Goals

Goals (or templates for goals) that occur regularly in certain types of applications can be formalized once and for all, so that the effort and the likelihood of design errors is reduced when they are included in specifications.

2.3 Trust Model

Another characteristic step in the design of a secure system is to determine a *trust model*. This is needed in addition to the usual specification because not every concerned party trusts all components, all steps of the design process, and all other users.

---

4 This can be made less likely by assigning priorities to individual goals [Brüg1_93]. Of course, the priorities must also be agreed upon. A usual case may be preformalized goals for classes of participants, which can be overruled by more specific goals.

Conversely, one often speaks of a model of *adversaries* or *attackers* or *threats*. By saying "trust", we want to stress three aspects:

- It is a relation, i.e., it depends not only on the component or person that is trusted or not, but also on the concerned party who trusts or not.
- The default value in the design of a secure system should be that something or someone is *not* trusted. One might not want to call someone an adversary or a threat just because they might possibly under certain circumstances attack, or just because a concerned party is not *forced* to trust that they will not.
- Both unintentional errors and purposeful attacks are meant.

Trust Subjects

In the real world, the subjects who express trust are the concerned parties. However, the trust sometimes depends on the individual goal. An important example is graceful degradation, where one has a series of weaker and weaker goals and needs less and less trust to achieve them. Hence, formally, the subjects in the trust model are just the *individual goals*.[5]

Note that goals of sets of concerned parties are formally treated just like those of individual parties, and so is their trust. However, in research areas with general rules to derive the trusted objects from the parties concerned, one needs rules for sets, too.

For a global specification that is not just a collection of individual goals, building in the trust model is more complicated: The subjects can only be the concerned parties, but even those are not mentioned in a normal specification. Hence one has to introduce them for this purpose, see the subsection "State-Oriented Global Service Specification" in Section 4.5.

Trust Objects

Usually, a concerned party can express under which circumstances it would trust components (to be designed later) by the following **trust criteria**:

- Who would design / implement / verify / test / ship the component? (And with how much care and which tools?)
- Where and under whose control would the component placed?
- What physical and organisational protection would there be for it?

Note that these trust criteria should be made for "basic" components, i.e., those that concerned parties trust or do not trust as a whole — combining such components into more trusted subsystems is another design stage.

Trust Degrees

In many cases, the trust model simply contains binary decisions whether certain objects are trusted for an individual goal or not. However, there may be more degrees of trust, described by a combination of statements of the following types:

---

5 Note that the trust model makes the individual design goals relevant formally, even if one has a global specification, too: The goals will have to be proved under these different trust assumptions.

- Worst possible behaviour of untrusted objects, e.g.:
  - One trusts that the computational resources of adversaries are limited, or that adversaries cannot break a so-called cryptologic assumption, e.g., factor 660-bit integers.
  - One trusts that data may be observed in a component, but the component does not deviate from its component specification.
- Characterization of sets of simultaneously trusted objects, e.g.:
  - One trusts at least *k* out of certain *n* people (such as designers of components).
- Quantification of trust, e.g.:
  - For the reliability of hardware components: parameters such as the mean time to failure.
  - For the trustworthiness of a physical protection measure: the price an adversary would probably have to pay to break it.

Additionally, trust may be a function of time.

2.4 Component Division and Component Specification

Given the specification and the trust model (and assuming one knows the semantics of those two together), one starts building a system. Among the many steps of such an implementation, (at least) one is particular to security, the **division into components** in the sense of the trust objects of Section 2.3: One determines the number of components, the trust criteria that each component fulfils, and how they are connected.[6] If the trust model offers the possibility to design and place a component so that it is trusted for all individual goals, one is lucky: One can implement the whole system in this component without further trust considerations and thus use all known design and verification techniques. (One may, of course, still prefer a decentralized solution for efficiency reasons.) Well-known strategies for other cases are to store certain data under the control of the respective party who has the primary interest in them, or to store copies in different components, or to distribute the data with a threshold scheme [Sham_79] so that they cannot be read by observing any single component.

Next, a specification of the **interface behaviour of the components** is needed. Strictly speaking, this has to be done in parallel with the previous step. In the area of formal verification of distributed systems, one often calls this the **protocol specification** in contrast to a service specification.

One can now try to **verify** that the system with these components fulfils the specification with respect to the trust model. Briefly, each individual goal has to be proved with only the given assumptions about the components. In particular, if subjects either trust objects completely or not at all, only the specifications of the former components can be used in the proof, whereas the other components can behave arbitrarily. Note that it is harder to see how a global specification is used with a trust model, see Section 4.5.

2.5 Implementation of Components

Finally, the components have to be implemented. This will usually involve a division into subcomponents and several design steps. However, as soon as one has a whole component in one trust domain (including the current designer and the tools) and the whole set of requirements on its interface behaviour, there is nothing security-specific to be considered.

---

6 Note that objects like networks, which have their own trust criteria, must be considered as components.

3 Known Research Areas in this Framework

We show how the best-known research areas within security fit into our framework. By this, we intend to show

- that all aspects of our framework are important in some known research areas,
- that our framework comprises a large class of (hopefully all) aspects of security, and
- how our notions relate to the terminology from different security communities, so that readers feel more familiar with them before we sketch how they may be formalized generally.

Note that most publications on formal security specification do not treat just one specification (e.g., in a normal specification language), but propose a special language to express a particular type of specifications for a particular type of systems.

3.1 Access Control and Information Flow Control

Access control has a strong aspect of relations between individual goals and global specifications:

- Individual goals: In the notation of [Denn_82], one considers “protection policies” that characterize certain states of a (virtual) access control matrix as permitted and others as forbidden; some states may be “don’t care”. If several concerned parties were considered, each of them would have their own wishes about states that should be forbidden or permitted.
- Global specification: What is called the actual specification consists of transition rules describing all the permitted state changes of the (virtual) access control matrix. It is complete by construction.

One tries to prove that the global specification fulfils the individual goals, i.e., all permitted states can be reached, but no forbidden one, and to automate such proofs for certain classes; many of the problems are undecidable, i.e., the classes are still too large [Denn_82].

Information flow specifications usually consist of individual goals, e.g., that no data flow should occur between certain variables or channels (for sequential programs and reactive systems, respectively) [Land_81]. There are detailed definitions of the semantics, i.e., what it means for a system (described in various protocol specification formalisms) to fulfil such a requirement, e.g., [GoMe_82, Brow_91, Gray_92]. (Access control specifications do not really have such a semantics, this is why they have some problems with “indirect access” in spite of a verification.)

Both with access control and information flow control, discussion of appropriate goals for certain situation belongs to the stage before a specification. Examples are the Bell-LaPadula model for secrecy in the military area and goals for systems handling personal data, including confidentiality in a “need-to-know” form, transparency, and the right of individuals to have incorrect data corrected [Land_81, BiBr_88]. Such goals lead to the design of languages for expressing the global specifications, (virtual) access control matrices or flow relations, in ways suited to these particular applications.

The usual trust model considers everything from the point of view of a system administrator as the only concerned party, and the whole system is considered trustworthy. (One does not trust the users, but that is already implicit in the specification.) There are a few publications about implementations in scenarios with smaller trust, e.g., about computing with encrypted data or access control by key hierarchies, e.g., [WaP1_86, ChTa_90].

3.2 Cryptology

A new branch of cryptology often starts with an informal sketch of a new type of systems with several concerned parties, e.g., untraceable communication or group cryptology [Chau_81, Desm_88].

Usually, one only has individual goals; in particular, the specifications are not complete. For instance, an encryption scheme should keep messages secret and a signature scheme should guarantee authenticity, but it is not fixed what else they do. (For instance, the definition permits that an encryption scheme may or may not guarantee authenticity, and a signature scheme may or may not guarantee secrecy.) Really formal specifications do not exist, in particular for large schemes, such as payment schemes — the goals are just definitions in terms of the structure of the implementation. An attempt at a structure-independent specification of signature schemes was made in [Pfit4_93].

An exception with global specifications and a nothing-else aspect is multi-party function evaluation; see Section 4.5.

The trust model of cryptology is very suspicious: One always assumes that some components are not trusted. The usual model is even that each set of concerned party only trusts its *own* components.[7] For instance, with encryption, the two communicating users only trust their end devices, not the network; and in secure payment systems [Cha8_85, Chau_89], the security that someone does not lose money only relies on the correctness of his own computer (often hand-held), not, e.g., on the goodwill of bank employees and programmers.

Sometimes one makes the stronger assumption that a certain percentage of the participants are trustworthy and their components correct, e.g., [Sham_79, RaBe_89].

3.3 Fault Tolerance and Safety

Fault tolerance mainly deals with trust models and the corresponding component division. However, one usually trusts that nobody is malicious — there are just physical faults and unintended design errors.[8] More formally, this means that errors are regarded as statistically independent (at least to a certain extent), and that one can make reasonable quantitative studies about appropriate trust models.

The concepts of fault tolerance are designed to work with any kind of specification (at least the usual non-security specifications). A specialty is that, for one and the same system, several sets of specifications for different trust models are often provided (graceful degradation).

Safety, in particular, deals with the design and verification of such fall-back specifications consisting of very few individual goals, mainly of things that should absolutely not happen.

---

7 Thus the trust of a set is usually the union of the individual trust, not the intersection. This is why the goals of sets cannot just be goals of each member of the set alone: All the components of the parties from the set must work correctly for the goal to be achieved.

8 An exception are articles like [PeSL_80], where one trusts a certain percentage of the participants and their components, but the rest may be malicious, just as in some parts of cryptology. For historic reasons, such schemes only count as cryptology if some privacy or confidentiality is required, whereas there are other branches of cryptology without privacy or confidentiality requirements, e.g., signature schemes.

3.4 Other Areas

Statistical databases are characterized by two types of individual goals: confidentiality goals, e.g., that no data should be reidentified, and goals that certain aggregated computations are possible. They are obviously conflicting goals of different concerned parties. The major research is about how one can determine if these goals are consistent, and one usually finds out that they are not.

Secure software usually just treats what is classically called correctness; sometimes, fault tolerance or safety aspects are included.

Risk analysis has aspects of the stage before a specification, e.g., strategies for discovering important goals, and aspects of quantitative analysis of trust models, e.g., the quality of physical and organizational measures protecting the digital components.

The analysis of covert channels can be seen as a quantitative information flow analysis on systems given by protocol specifications at a low level.

The design of secure operating systems has a trust model where one trusts designers as little as possible (both with respect to attacks and to unintentional faults); therefore, one concentrates on small security kernels.

4 Overview of Formalizations

Formal notions of security centre on specifications and their semantics, i.e., the definition of what it means for a system to *fulfil* a specification securely, in particular, when a trust model is given. The following are our main theses for a general formal treatment (none of them quite new in itself, but different from informal publications about general security notions, at least in this combination). We hope to substantiate these theses by the following subsections.

- One has to regard specifications as descriptions of **interface** behaviour. Given a non-trivial trust model, one has to distinguish an **intended** and an **altered** interface; then even rather elaborate semantics like in [GoMe_82, Jaco_88] must be extended.
- It is useful to distinguish two big classes of properties, **service** properties and **ignorance** properties.[9] (Why we are so sure is explained in the subsection "Individual Ignorance Goals" of Section 4.5.)
- It is helpful to regard non-security research on specifications, in particular, of **reactive** systems (i.e., systems that may react with their environment more than once, in contrast to sequential programs, e.g., [MaPn1_91]). Many aspects of security are then new types of semantics for conventional specifications rather than new types of specifications.
- In particular, it is useful to distinguish **property-oriented** and **state-oriented** (or descriptive versus constructive) specifications: In most cases, individual goals are formulated in a property-oriented way and global specifications in a state-oriented way.

---

[9] "Knowledge properties" sounds better, but the requirement is that someone does *not* know something.

4.1 Interfaces

By **interface**, we mean the complete set of connections between the system and the outside world. We use the word as in the OSI architectural model and often distinguish different access points to the system, too [Lini_83].[10] This is illustrated in Figure 2.

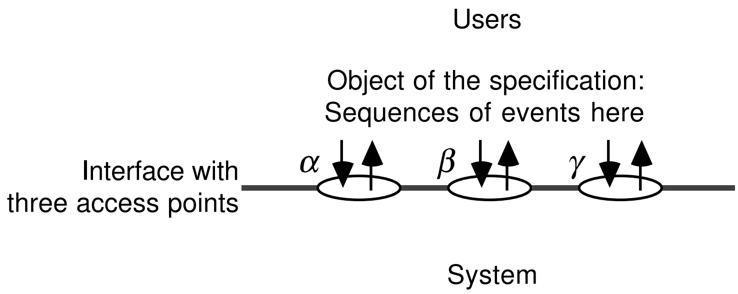

Figure 2 Illustration of the notions of interface and access points

Formally, such an interface is just a set, called the set of **access points**. (Thus it is $\{\alpha, \beta, \gamma\}$ in Figure 2.)

With non-trivial trust models (more precisely, whenever one cannot find globally trusted components to be connected to all parts of the interface from within the system), one will need two variants of the interface:

- the intended interface, and,
- given the trust for a certain goal and a system, the altered interface between the trusted part of the system and its environment, which includes the untrusted parts. In other words, adversaries usually extend the interface, e.g., by tapping lines.

The two interfaces are illustrated in the following two figures: Figure 3 shows the intended interface with a system beneath; the system is given by the component specification of the intended components.

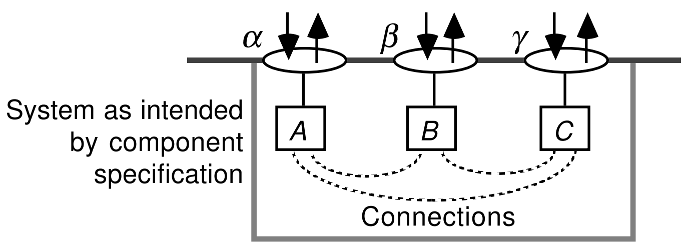

Figure 3 Intended interface and intended components of a simple distributed system

Figure 4 shows the same system and its altered interface from the point of view of a certain set of concerned parties (e.g., because one of their individual goals is currently considered).

---

[10] In the simplest case, each access point is the user interface of a different computer in a distributed system; this is the standard case in cryptology. But different access points can also be implemented in software on one computer, e.g., the different views a time-sharing system shows its users according to the initial user identification.

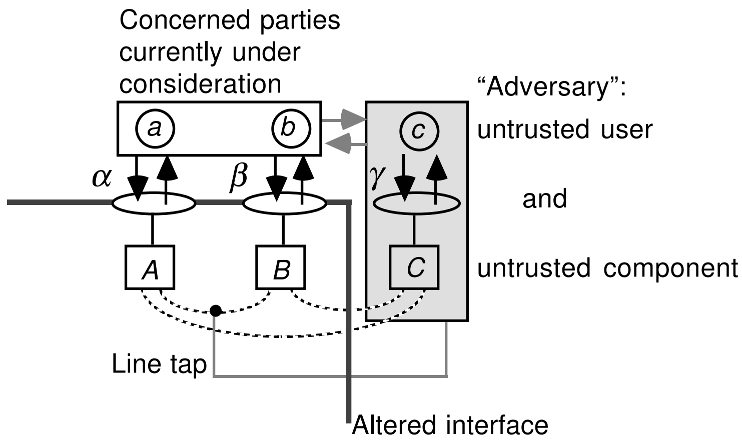

Figure 4 Example of an altered interface.
The system from Figure 3 is shown from the point of view of the concerned parties *a* and *b*, who do not trust *c* and the component *C* and fear that the line between their components *A* and *B* may be tapped.
Note that the altered interface is different for every other set of concerned parties, assuming they all trust their "own" local components only.

So far, we considered the altered interface for a given system. During the specification, we cannot refer to the complete altered interface yet. However, one can define the **interface to the concerned parties** as an arbitrary subset of the access points. In Figure 4, it is $\{\alpha, \beta\}$, i.e., the horizontal part of the altered interface. (This set will be used to indicate that the concerned parties themselves interact with the system at these access points, and one has different goals on what happens here than on what happens with the adversaries.)

4.2 Behaviour at an Interface

Arguably the most central notion in any treatment of specification and verification is the formal model of the behaviour of a system at an interface. (A specification will then be a *description* of one or more permitted behaviours, where description means by a string of finite length.)

In any notion of behaviour of reactive systems one can somehow identify the basic notion of a **run** of a system.[11] In digital systems with discrete time, a run is seen at the interface as a **sequence of interface events**, i.e., inputs and outputs at individual access points. Note that the restriction to

- digital systems with
- discrete time and
- discrete access points,

although very common, is security-critical, at least as far as the access points of adversaries are concerned: In reality, one may have overlooked possibilities of access, and the adversaries can observe real-time changes of physical quantities such as voltage. Hence whenever the digital values that should only change at discrete times are imperfectly realized, adversaries have more possibilities to draw conclusions than in the model.

11 Related terms are *execution* and *trace*.

Moreover, to be specific, we restrict ourselves to synchronous systems, i.e., systems with a notion of rounds where inputs and outputs occur. Then the sequence of interface events is indexed by the rounds, and there may be several events or no event at all in each round.[12]

The **interface behaviour** of a system is now a model of the totality of the possible runs.

- The simplest formal representation of "totality" is the **set** of the possible runs. This is called a linear-time model, and it is sufficient for many important purposes. (See, e.g., [MaPn1_91].)
- More elaborate models retain some idea of the internal state of the system in the totality of runs. In particular, for probabilistic systems, one can consider the sequence of transition probabilities from the sequence of inputs and outputs up to a certain round to the next output (e.g., as in [Gray_92]).[13]

4.3 Service and Ignorance

Specifications will be descriptions of (sets of) permitted interface behaviours. If we have a trust model where the interface can be altered, it is unreasonable to expect that we can exactly prescribe any behaviour that could be achieved in the case with global trust. E.g., in Figure 4, it makes no sense to prescribe the outputs at the access point γ precisely, since we cannot prevent component C from outputting arbitrary values.

Hence, for security, in contrast to normal specifications, we have two rather clearly distinguished types of properties that *can* reasonably be required:

- What we call **service properties** is restrictions on the outputs that the system gives to the concerned parties, i.e., at the interface to the concerned parties. These properties comprise what is usually called integrity and availability.
- What we call **ignorance properties** is restrictions on the outputs that the system gives to the adversaries, but only in the sense that the knowledge or information that can be inferred from these outputs is restricted. In particular, we consider the adversaries' knowledge or information about the events at the interface to the concerned parties.[14]

Note that combined properties exist, and in fact, ignorance is usually considered in combination with some service property.

---

[12] The best-known alternative is that there are no rounds, and the i-th element of the sequence is the i-th event that occurs overall. Then one usually does not consider simultaneous events, but assumes that they are ordered somehow (interleaving). However, for security purposes, such a specification must certainly be refined to one that includes the real time of the events, because adversaries may draw conclusions from times.

[13] For non-deterministic systems, which we are not considering in the following, there are models with trees (branching-time models; e.g., [Emer_90]), where runs are on the same branch of the tree as long as they correspond to the same internal state. One can also assign probabilities to branches of such trees.

[14] We used the strange term "ignorance" because we hope that it cannot be misunderstood. Related terms that are sometimes used with the same meaning are confidentiality (but some authors restrict this to confidentiality of message contents, and not, e.g., anonymity), privacy, secrecy, and uncertainty.

More About Service Properties

Service properties are the usual type of properties, as known from most specification styles outside security. In a linear-time model, a service property is just a set of permitted sequences of interface events.

In our case with intended and altered interfaces, a pure linear-time service property is a set of permitted sequences of events at the interface to certain concerned parties. A pure linear-time **service specification** is therefore a string *spec* that describes such a set. More precisely, *spec* must be an element of a (specification) language *L*, and there must be a mapping *Sem* from *L* to such sets of sequences, called the natural semantics of the language[15].

Moreover, not any such set will do, because a specification should only specify the system, not its users. Hence it cannot exclude inputs, it can only specify

- that the system does not react on them — this is the formal equivalent of forbidding a user to perform certain operations — or
- that the system can, from then on, behave arbitrarily — this is the formal equivalent of requiring honest users to observe certain rules if they want the system to protect them. (However, there is the notion of **robustness** that says that the specification should be like this as little as possible.)

More About Ignorance

What we call ignorance properties is distinguished from service properties by the concept of **alternative "worlds"**, i.e., the totality of runs that could have occurred from the point of view of certain participants.

Outside security, there is a "logic of knowledge" (a special kind of modal logic, see [Halp_87]) with basically this semantics: Someone does not know something if there is an alternative "world" that she cannot distinguish from the real one, and where this fact is not true. In our case, the "worlds" are actual runs of the system, and an adversary cannot distinguish two behaviours if they are equal at those parts of the interface that the adversary can observe.

However, this is deterministic knowledge, i.e., absolutely certainty about something. It has been used in security, too. More often, however, one requires the absence of partial knowledge. This is information theory, where the possible worlds are assigned probabilities (e.g., see [Sha1_49, Gray_92] for encryption schemes and information flow in this model.) There are currently almost no general specification techniques for information-theoretic properties, just direct definitions in terms of the system structure — this makes cryptologic definitions so error-prone.[16]

Integrity and Availability, and Other Types of Properties

If we try to formalize the difference between integrity and availability at the specification level, they become just what is called **safety** and **liveness** in the specification community [AlSc_85].

---

15 "Natural" in contrast to the security semantics sketched below.

16 Cryptology often assumes computationally restricted adversaries; this yields modifications to the information-theoretic notion of ignorance. For brevity, we cannot treat this in detail. The main idea is that the adversary component (i.e., the gray component in Figure 4) has to make outputs that explicitly represent what the adversary has learnt. For instance, it has to distinguish by this output which of a certain class of events occurred.

With this distinction of integrity, availability, and ignorance ≈ confidentiality, we are quite near to classical views of basic classes of security properties. Recently, it has been argued that more classes are needed, such as **authenticity**, **non-repudiation**, and **anonymity**. We support that these are important properties, and not well covered in classical treatments like the example functionality classes in [ITSEC2_91]. Formally, however, we think they are special service and ignorance properties, just as the classical restrictions (e.g., to confidentiality of file contents) are only special cases.

Completeness

Given a certain interface, we will call a specification complete iff it describes exactly one interface behaviour. (See Section 4.5, “Ignorance as Additional Semantics of a State-Oriented Global Service Specification” for how we intend to apply this notion in the scenario with intended and altered interfaces.) With a linear-time model, this means that the specification precisely determines the outputs (as a function of all the previous inputs). With behaviour described by transition probabilities, this means that all the transition probabilities to outputs are fixed.

This notion of completeness is one reason why we emphasize that specifications only refer to interface behaviour — if a similar notion were applied to internal states, it would mean that the specification completely fixes the internal states of the implementation. Moreover, we think that this notion of completeness is the only one where one can reasonably define a nothing-else aspect for, see below. However, as long as there is just one interface, completeness even in this sense implies a nothing-else property automatically: If there is no freedom in the interface behaviour, the system cannot do anything else there anyway.

4.4 Property- and State-Oriented Specification Techniques

Most treatments of specification techniques distinguish the following most important classes:

- Property-oriented or descriptive specifications define properties directly, e.g., all sorts of logics (such as normal predicate logic, temporal logic [Emer_90, MaPn1_91], and logic of knowledge [Halp_87]).
- State-oriented or constructive specifications primarily describe a system, usually by states and transition rules. The behaviour of this system is regarded as a **reference** for the intended behaviour of the real system.

  Outside security, this is only done with respect to service, i.e., the interface behaviour of the real system is compared with that of the reference system. This is particularly useful if the reference system can be simpler than the real system, e.g., by using a more powerful language. In particular, the service specifications of a distributed system may be a centralized and therefore simple reference system, as in Figure 5.

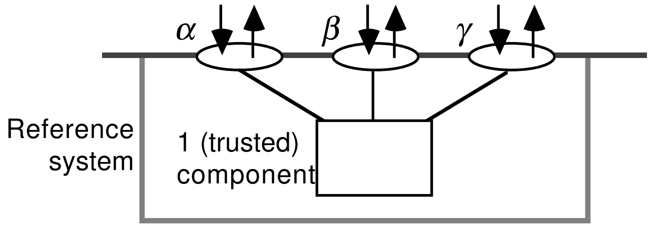

Figure 5 Example of a constructive specification: A centralized reference system for the interface in Figure 2 (and thus suitable for specifying the system in Figure 3).

Examples of constructive specification languages are extended finite state automata and derived languages like Estelle [ISO 9074_89], process algebras [Miln_80], and Petri nets [Pete_77], or, within security, access control matrices with transition rules.

Property-oriented specifications are quite natural for individual goals (since one can combine several of them easily), state-oriented ones for global specifications. Moreover, most of the latter are naturally complete. However, intermediate cases exist, such as a completion of individual goals by prescribing initial semantics [DoYa_83, EhMa_85].

State-Oriented Specification and Interfaces

Note that we intend even state-oriented specifications to *refer* only to the interface behaviour of the system, i.e., we do not require the internal states of an implementation to be identical to those in the specification. One reason was mentioned in Section 4.3, "Completeness". Another reason is that only this semantics allows cryptologic implementations: In a cryptologic implementation of any type of system (more generally: whenever a complicated trust model is given), the internal states are naturally much more complicated than those of the specification.

For instance, a secure information system may be specified by a so-called mini-world consisting of relations with possible update and query operations and rights on such operations. In a distributed implementation where not all nodes are trusted by everybody, so that data are encrypted and secret sharing is applied, the relations as specified will not be found anywhere in the system. Nevertheless, one can define that such an implementation is correct if any sequence of updates and queries yields the same outputs as it would in the specified mini-world.[17,18]

---

17 One might argue that a mapping from the real data to the abstract mini-world relations will exist, which should be considered instead of the interface behaviour. However, to define generally what a correct mapping is, one will end up comparing interface behaviours again.

18 This should even apply to simpler implementations where the components of the specification exist, but in refined form. This is why one has problems with "indirect access" in systems proven secure with respect to an access matrix specification (in contrast to an information flow specification): Such a proof is state-oriented — correctness of the reference monitor —, but one had an intuitive interface interpretation of the specification, which can still be violated.

4.5 Semantics with Trust Models

The main new aspect when specifications are given a semantics in security is the trust model, and in particular, the two different interfaces:

- The specification should be in terms of the intended interface (as in Figure 2), because the altered interface is only known when the design process has advanced to the component division. Similarly, the specification of components in the sense of Section 2.4 describes intended components (as in Figure 3).
- The semantics, however, must deal with the altered interface and consider that some components are not correct (as in Figure 4).

The basic problem is that if an incorrect component serves a part of the intended interface, the component may be completely dismantled, or at least it cannot be prevented from making other interface outputs than the correct component. This may sound absurd and insecure to people from some fields, but it is the standard case in cryptology. Recall that the system is regarded from the point of view of different concerned parties, and it only has to be ensured that if *you* don't trust a component, it does not influence those parts of the interface that occur in *your* goals. For instance, if many people use the same home banking system as you, you don't care if a hacker manipulates his terminal so that it can no longer make payments, as long as you can.

To say more precisely what it means for *any* system to fulfil *any* specification with such a trust model, it is useful to distinguish the four important specification types (property- and state-oriented service and ignorance specifications). Note that all four occur in the security literature, at least informally or for specific systems, and we believe that they are useful, but we can identify a rigorous meaning for some of them quite satisfactorily, whereas others leave more open questions.

Individual Service Goals

The semantics of individual linear-time service properties with a non-trivial trust model is the clearest. One just has to make an introductory requirement that

- the service property is only about the interface to the concerned parties for this goal, and
- after the component division, all parts of the interface to the concerned parties are served by components trusted by these parties.

This requirement is quite natural: The usual case where components that are not globally trusted serve any part of the interface is that different concerned parties have different access points, because the system is distributed, and the parties do not trust all remote computers (see Figure 4). Then a service (!) goal of a set of parties naturally only says what service the system should offer at the access points of these parties — in correct relation to each other, of course. (In the example of Figure 4, this requirement is fulfilled if the goal of the users a and b only deal with what happens at the access points α and β. An example goal is that b obtains data entered by a in unmodified form.)

Given that this introductory requirement is fulfilled, the semantics of the individual service goal is that

- for *all* strategies[19] of the adversaries and
- for *all* user strategies (which may interact with the adversaries outside the system, as indicated by the gray arrows in Figure 4),

the resulting behaviours of the system at the interface to the concerned parties are permitted according to the goal.

In Figure 4, this means that we universally quantify over the behaviours of the big gray and white rectangles, and only the components A and B act according to the component specification. Note that as long as there is no restriction on the interaction between the concerned parties and the adversaries outside the system, this is equivalent to a model where the trusted components interact with one big adversary strategy, i.e., the gray and the white rectangle are joined.

This approach has been carried out in detail for specifications of different new types of digital signature scheme in [Pfit4_93]. It should cover all other authentication problems, too, and also almost all pure integrity and availability properties ever mentioned in the literature. Probable exceptions are bit commitment schemes [BrCC_88], which seem to need a branching-time specification if one considers them at this level, and the integrity aspects of schemes like election schemes, which require independence of simultaneous inputs of different participants. Those, however, are usually treated with global, state-oriented specifications anyway.

State-Oriented Global Service Specification

For simplicity, we only discuss *state-oriented* global service specifications.

Such a specification must be in terms of the intended interface (since the other interfaces vary with the individual goals), and will mention all parts of the interface (cf. Figure 5). Hence a security semantics of such a specification under a non-trivial trust model is not as simple as that of an individual goal. One has to compare the restrictions of the specified and the real behaviours to the interface to the concerned parties. (Hence these “interfaces to concerned parties” yield the formal trust subjects for global specifications, see Section 2.3.) The notion of “restriction” is not trivial: Clearly, only outputs to concerned parties are compared, but they may legally depend on inputs from untrusted parties (e.g., the result of a voting protocol), whose interface part has disappeared in the real system. Hence one defines that real outputs are correct if there exists a virtual adversary input strategy that would lead to these outputs in the reference system.

For example, if the system of Figure 4 is compared with the specification of Figure 5, one compares with the virtual adversary illustrated in Figure 6 and requires that for all strategies of the adversary in Figure 4, there exists a strategy for c^* that has essentially the same effect on the concerned parties.

---

19 We use “strategy” similar to game theory. Formally, a strategy of an adversary is just like a protocol of a correct component, but we wanted to avoid confusion with the protocol that the incorrect components *should* have carried out according to the protocol specification.

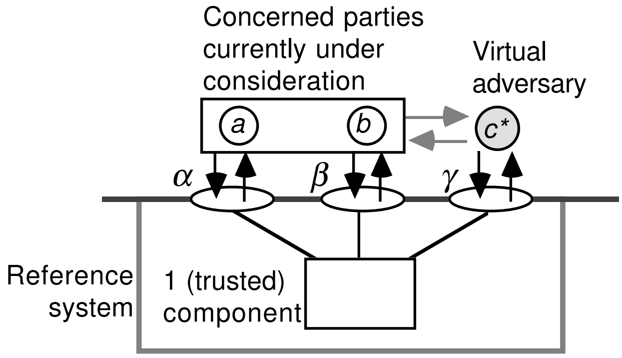

Figure 6 Treating adversary inputs in the security semantics of a state-oriented global service specification.
The system in Figure 4 is defined to be a secure implementation of the specification in Figure 5 iff for each strategy of the adversary in Figure 4, a strategy for a virtual user c^* exists that has essentially the same effect at the access points α and β. (All concerned parties have to be treated like this in turn.)

For specific examples, see [Beav5_91, MiRo1_91]; they are explained in more detail below, since they include ignorance.

Individual Ignorance Goals

An individual ignorance goal of certain concerned parties expresses that *other* users and adversaries in untrusted components do not gain certain knowledge, usually about previous interface inputs of the concerned parties. This is what makes ignorance goals so fundamentally different from service goals: They have to do with the *untrusted* part of the interface. Recall that it makes no sense to require any service properties for that part (at least if a component is completely untrusted), but you can require and fulfil ignorance goals, i.e., you can prescribe that in spite of arbitrary behaviour of the untrusted part, the observations leave the required alternatives as to what happened in the trusted part. In the example of Figure 4, an ignorance goal of the concerned parties would treat what can be learned about the horizontal part of the altered interface, and the semantics evaluates this according to the vertical part.

No general specification language and definition for this case exist, but many direct definitions for various special cases, e.g., secrecy definitions of encryption schemes in cryptology [Sha1_49, MiRS_88, RaSi_92] and definitions of the absence of information flow [Brow_91, Gray_92]. Note that the complete absence of information flow between certain access points, which is sometimes defined as security as such, is not suitable for all ignorance requirements. For instance, with an electronic payment system such as [Cha8_85, Chau_89], everybody can exchange payments (and thus information, e.g., in the amount and time of the payment) with everybody else. Nevertheless, one wants to guarantee that nobody learns anything about the payments of others (unless they divulge this information deliberately, and this possibility makes general definitions hard [Schu_94] — note that this concerns possible channels both through and outside of the system).

Ignorance as Additional Semantics of a Complete Service Specification

The other way of specifying ignorance is to use a complete service specification (for simplicity again assumed to be state-oriented, and deterministic or probabilistic), and to add ignorance to the semantics only: Such a semantics requires that the real system does “nothing else” than the reference system; in particular, it does not give an adversary more information than the reference system would.

Without using terms like “specification” and “semantics”, this approach was suggested for general definitions in cryptology in [Yao_82]. The problem treated there was that each of n participants has a secret value x_i, and they want to compute $y := f(x_1, \ldots, x_n)$ for a given function f. However, nobody should gain more information about the inputs of the others than what can be deduced from the result y alone. Many articles about suitable definitions of this concept followed, e.g., [Beav5_91, MiRo1_91]. In our terms, this is a global specification of a non-reactive system by a (trusted) reference system that gets the inputs x_i and computes and outputs y.

Essentially, the semantics that a real distributed protocol, where no individual party is trusted, does “**nothing else** than this reference system” is:

- For all sets of concerned parties,
- for all cheating strategies that the untrusted components and other users can use in the real protocol,

there is another strategy they could use with the reference system that yields both

- the same behaviour at the interface to the concerned parties, and
- the same knowledge for the adversaries.

In the example of Figures 4 to 6, we would need an additional output at both adversaries (i.e., the gray rectangle and the gray circle), which models the knowledge gained (similar to Footnote 16). Thus, it is not only modeled that the adversary learns nothing else about the behaviour of the concerned parties, but “nothing else” as such. This is represented by requiring that everything that the real adversary can output, the virtual adversary on the reference system can output just as well. (Such properties are known as **minimum-knowledge**, with the special case of zero-knowledge [GoMR_89].)

Since some details of this approach are still disputed in the given non-reactive case, it is certainly not trivial to extend it to the reactive case, but we think it is worth while.

5 Outlook

Of course, in such a brief informal sketch of the framework, a lot of details were omitted. Some are listed here:

- We did not regard probabilistic or branching time individual service goals very much.
- For most applications to cryptology, one has to allow
  - small error probabilities, or
  - computational restrictions (on the adversary strategies and user behaviours over which one has the universal quantification),

  or both in the semantics. (The reader is encouraged to look at [MiRS_88, Beav5_91, Pfit4_93].) Formally, this means that there is not just one notion of “fulfilling” a specification, but the specification can be fulfilled with different **degrees of security**. (If one starts with a standard view of what a specification is, the trust model can also be seen as a part of the degree of security, and so can some restrictions on the behaviour of the concerned parties.) Both for error probabilities and computational restrictions, one has to consider a sequence of systems, depending on a "**security parameter**" (or more than one), all with the same interface. Some details for individual service goals can be found in [Pfit4_93], some more in [Pfit9_93].

- The access of users to access points has to be restricted, e.g., by organizational and physical measures, or by passwords or biometrics. This is fairly standard [DaPr_89], unless in combination with anonymity, such as in [Cha8_85].

Other questions are open, e.g., a general way of handling what a user divulges "deliberately" (see Section 4.5).

Moreover, we have not treated refinement (in particular, of interface actions) nor proof techniques.

- Refinement will certainly have to distinguish the part of the interface belonging to the concerned parties whose goal (or restriction of a global specification) is currently considered — they can be assumed to use the system as prescribed — and the rest.
- Proof techniques can be distinguished into those where the system to be proved secure is described in a standard protocol specification language, and those that treat combinations of subsystems that are only known to be secure with some degree of security into larger systems.

However, all this can only come *after* a precise semantics of what security under trust models and with different degrees of security means.

Acknowledgments

A paper of such generality has naturally profited from discussions with a large number of people. In particular, many of our ideas have developed during joint work with *Andreas Pfitzmann*. We also thank *Birgit Baum-Waidner*, *Eike Best*, *Joachim Biskup*, *Gerrit Bleumer*, *Jimmy Brüggemann*, *Ulla Goltz*, *Winfried Görke*, *Michaela Huhn*, *Peter Niebert*, and *Matthias Schunter* for discussions on various aspects of security or specification in general.

References

AlSc_85 Bowen Alpern, Fred B. Schneider: Defining Liveness; Information Processing Letters 21 (1985) 181-185.

Beav5_91 Donald Beaver: Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority; Journal of Cryptology 4/2 (1991) 75-122.

BiBr_88 Joachim Biskup, Hans Hermann Brüggemann: The Personal Model of Data; Computers & Security 7/6 (1988) 575-597.

Bisk_93 Joachim Biskup: Sicherheit von IT-Systemen als "sogar wenn – sonst nichts – Eigenschaft"; Proceedings GI-Fachtagung VIS '93; DuD Fachbeiträge 16, Vieweg, Wiesbaden 1993, 239-254.

Brow_91 Randy Browne: The Turing Test And Non-Information Flow; 1991 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, Washington 1991, 373-385.

Brüg1_93 Hans H. Brüggemann: Prioritäten für eine verteilte, objekt-orientierte Zugriffskontrolle; Gerhard Proceedings GI-Fachtagung VIS '93; DuD Fachbeiträge 16, Vieweg, Wiesbaden 1993, 51-66.

BuAN_90 Michael Burrows, Martin Abadi, Roger Needham: A Logic of Authentication; ACM Transactions on Computer Systems 8/1 (1990) 18-36.

Chau_81 David Chaum: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms; Communications of the ACM 24/2 (1981) 84-88.

Cha8_85 David Chaum: Security without Identification: Transaction Systems to make Big Brother Obsolete; Communications of the ACM 28/10 (1985) 1030-1044.

Chau_89 David Chaum: Privacy Protected Payments – Unconditional Payer and/or Payee Untraceability; SMART CARD 2000: The Future of IC Cards, Proceedings of the IFIP WG 11.6 International Conference; Laxenburg (Austria), 19.-20. 10. 1987, North-Holland, Amsterdam 1989, 69-93.

ChTa_90 Gerald C. Chick, Stafford E. Tavares: Flexible access control with master keys; Crypto '89, LNCS 435, Springer-Verlag, Heidelberg 1990, 316-322.

DaPr_89 Donald W. Davies, Wyn L. Price: Security for Computer Networks, An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer; (2nd ed.) John Wiley & Sons, New York 1989.

Denn_82 Dorothy Denning: Cryptography and Data Security; Addison-Wesley Publishing Company, Reading 1982; Reprinted with corrections, January 1983.

Desm_88 Yvo Desmedt: Society and group oriented cryptography: a new concept; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 120-127.

Dier2_91 Rüdiger Dierstein: The Concept of Secure Information Processing Systems and Their Basic Functions; IFIP/Sec '90, Helsinki; North-Holland, Amsterdam 1991, 133-149.

DoYa_83 Danny Dolev, Andrew C. Yao: On the Security of Public Key Protocols; IEEE Transactions on Information Theory 29/2 (1983) 198-208.

EhMa_85 H. Ehrig, B. Mahr: Fundamentals of Algebraic Specifications 1; EATCS Monographs on Theoretical Computer Science Vol. 6, Springer-Verlag, Berlin 1985.

GoMe_82 J. A. Goguen, J. Meseguer: Security Policies and Security Models; 1982 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Washington 1982, 11-20.

GoMR_89 Shafi Goldwasser, Silvio Micali, Charles Rackoff: The Knowledge Complexity of Interactive Proof Systems; SIAM J. Comput. 18/1 (1989) 186-207.

Gray_92 James W. Gray III: Toward a Mathematical Foundation for Information Flow Security; Journal of Computer Security 1/3,4 (1992) 255-294.

Halp_87 Joseph Y. Halpern: Using Reasoning about Knowledge to Analyze Distributed Systems; Ann. Rev. Comput. Sci. 1987, 37-68.

ITSEC2_91 European Communities - Commission: ITSEC: Information Technology Security Evaluation Criteria; (Provisional Harmonised Criteria, Version 1.2, 28 June 1991) Office for Official Publications of the European Communities, Luxembourg 1991 (ISBN 92-826-3004-8).

Jaco_88 Jeremy Jacobs: Security Specifications; 1988 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Washington 1988, 14-23.

Land_81 Carl E. Landwehr: Formal Models for Computer Security; ACM Computing Surveys 13/3 (1981) 247-278.

Lini_83 Peter F. Linington: Fundamentals of the Layer Service Definitions and Protocol Specifications; Proceedings of the IEEE 71/12 (1983) 1341-1345.

MaPn1_91 Zohar Manna, Amir Pnueli: The Temporal Logic of Reactive and Concurrent Systems: Specification; Springer-Verlag, New York 1991.

Miln_80 Robin Milner: A Calculus of Communicating Systems; LNCS 92, Springer-Verlag, Berlin 1980.

MiRo1_91 Silvio Micali, Phillip Rogaway: Secure Computation (Chapters 1-3); Laboratory for Computer Science, MIT, Cambridge, MA 02139, USA; distributed at Crypto '91.

MiRS_88 Silvio Micali, Charles Rackoff, Bob Sloan: The Notion of Security for Probabilistic Cryptosystems; SIAM J. Comput. 17/2 (1988) 412-426.

Mund_93 Sibylle Mund: Sicherheitsanforderungen – Sicherheitsmaßnahmen; Proceedings GI-Fachtagung VIS '93; DuD Fachbeiträge 16, Vieweg, Wiesbaden 1993, 225-237.

PeSL_80 Marshall Pease, Robert Shostak, Leslie Lamport: Reaching Agreement in the Presence of Faults; Journal of the ACM 27/2 (1980) 228-234.

Pete_77 James L. Peterson: Petri Nets; ACM Computing Surveys 9/3 (1977) 223-252.

Pfit4_93 Birgit Pfitzmann: Sorting Out Signature Schemes; 1st ACM Conference on Computer and Communications Security, 3.-5.11.1993, Fairfax, acm press 1993, 74-85.

Pfit9_93 Birgit Pfitzmann: Fail-Stop Signature Schemes; eingereicht als Dissertation, Institut für Informatik, Universität Hildesheim, 4.9.1993.

PfWa2_92 Birgit Pfitzmann, Michael Waidner: How to Break and Repair a "Provably Secure" Untraceable Payment System; Crypto '91, LNCS 576, Springer Verlag, Berlin 1992, 338-350.

PWP_90 Birgit Pfitzmann, Michael Waidner, Andreas Pfitzmann: Rechtssicherheit trotz Anonymität in offenen digitalen Systemen; Datenschutz und Datensicherung DuD 14/5-6 (1990) 243-253, 305-315.

RaBe_89 Tal Rabin, Michael Ben-Or: Verifiable Secret Sharing and Multiparty Protocols with Honest Majority; 21st Symposium on Theory of Computing (STOC) 1989, ACM, New York 1989, 73-85.

RaLa_92 Brian Randell, Jean-Claude Laprie: Predictably Dependable Computing Systems: Final Report; PDCS Technical Report Series No. 62, July 1992; Bulletin of the European Association for Theoretical Computer Science (EATCS) 48, Oktober 1992, 55-87.

RaSi_92 Charles Rackoff, Daniel R. Simon: Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack; Crypto '91, LNCS 576, Springer Verlag, Berlin 1992, 433-444.

Schu_94 Matthias Schunter: Spezifikation von Geheimhaltungseigenschaften für reaktive kryptologische Systeme; Diplomarbeit, Universität Hildesheim, February 1994.

Sham_79 Adi Shamir: How to Share a Secret; Communications of the ACM 22/11 (1979) 612-613.

Sha1_49 C. E. Shannon: Communication Theory of Secrecy Systems; The Bell System Technical Journal 28/4 (1949) 656-715.

Syve1_93 Paul F. Syverson: Adding Time to a Logic of Authentication; 1st ACM Conference on Computer and Communications Security, 3.-5.11.1993, Fairfax, acm press 1993, 97-101.

WaP1_86 Neal R. Wagner, Paul S. Putter, Marianne R. Cain: Encrypted Database Design: Specialized Approaches; IEEE Symposium on Security and Privacy, 1986, 148-153.

Yao_82 Andrew C. Yao: Protocols for Secure Computations; 23rd Symposium on Foundations of Computer Science (FOCS) 1982, IEEE Computer Society, 1982, 160-164.