HILDESHEIMER
INFORMATIK-
BERICHTE
ISSN 0941-3014

Birgit Pfitzmann

Sorting Out Signature Schemes

4/93 (Mai 1993)

UNIVERSITÄT
HILDESHEIM

Dieser Bericht ist herausgegeben vom

Institut für Informatik

Postfach 10 13 63
31113 Hildesheim

Sorting Out Signature Schemes

– and some Theory of Secure Reactive Systems –

Birgit Pfitzmann
Institut für Informatik, Universität Hildesheim,
Samelsonplatz 1, W-3200 Hildesheim, FRG

Abstract. Digital signature schemes are a fundamental tool for secure distributed systems. A formal definition of digital signature schemes is important, so that there is a clear interface between designers and users of such schemes. A definition that seemed final was given by Goldwasser, Micali, and Rivest in 1988. Recently, however, several signature schemes with new security properties have been presented. Most of them exist in several variants, and some of them pay for the new properties with restrictions in other respects. These new properties need definitions and some classification. Unfortunately, however, none of the new schemes is covered by the definition mentioned above. Hence the new properties cannot be defined as additions, but each new type of scheme needs a new definition from scratch. This is unsatisfactory.

This paper presents a general definition of digital signature schemes that covers all known schemes, and hopefully all that might be invented in future. Additional properties of special types of schemes are then presented in an orthogonal way, so that existing schemes can be classified systematically.

It turns out that signature schemes are best defined by a separation of service, structure, and degree of security, with a service specification in temporal logic. Several parts of such a definition can easily be reused for general definitions of other classes of reactive cryptologic schemes, such as payment schemes.

Relations to secure multi-party protocols and so-called logics of authentication are discussed.

1 Introduction

The purpose of digital signature schemes is to provide message authentication such that disputes about the authenticity of messages can be settled by third parties, such as courts. Digital signature schemes are therefore needed whenever messages of legal relevance are exchanged over communication networks, and they are also an important tool in distributed secure systems. As with all security critical systems, it is important to have a formal definition of what a signature scheme is, so that one can separate the design and the application of such schemes. Applications should work with any digital signature scheme that fulfils the definition, without caring about internal details, and designers should know what properties are expected from their schemes.

1.1 Previous Definitions

Although signature schemes are rather "small" schemes, e.g., in comparison with secure operating systems or, within cryptology, general multi-party computation protocols, agreeing on a definition has not been trivial. The first informal definition by Diffie and Hellman saw signature schemes just as a form of encryption schemes used in an inverse way, but nowadays many schemes, e.g., the proposed NIST digital signature standard, are not of this form [DH76, DSS92]. Moreover, it was originally overlooked that adversaries might try to compute new signatures from given signatures, in particular with RSA [RSA78, D84]. Hence it was very satisfactory that a definition that seemed final was given in [GMR88]. It allows the system to consist of an arbitrary key generation algorithm, an arbitrary signing algorithm, and an arbitrary test algorithm. Hence it seemed that all signature schemes would be of this form, in contrast to the Diffie-Hellman definition. Moreover, a strong security definition covering all conceivable behaviors of an adversary was given.

The GMR definition is important in practice even though most schemes in actual use cannot be proved secure with respect to it: They are all designed so that one can at least hope that they fulfil it, e.g., by the inclusion of hash functions or redundancy to counter active attacks, such as in the proposed NIST digital signature standard or in ISO standardization activities [DSS92, GQ91, ISO91]. Note that hash functions or redundancy have to be considered parts of the signature scheme if one wants to provide a common interface to applications. Also note that the most general active attack, where the adversary can ask the signer to sign arbitrary messages before it tries to produce a forgery, seems a bit excessive, but no restricted form is known where one is fairly sure to be on the safe side: It is unlikely that a signer will sign *any* message, but *what* message could one be sure she would not sign? Hence one does in fact take measures against the general attack in practice.

1.2 The Challenge of New Schemes

Just after this theoretically satisfying state with a generally agreed definition was reached, however, several new types of signature schemes with new security properties came up. For instance, there are *fail-stop* signature schemes, where a signer who has been cheated with a forged signature can prove that it was a forgery [BPW91, P91, HP92, HPP92]. A different example are so-called *undeniable* signature schemes [CA90, C91, BCDP91]. Here the recipient of a signature cannot show it around, e.g., to his friends, without the help of the signer. (Hence they are also called “invisible” sometimes — not being deniable is common to all types of signatures.) This property can be useful if it should be private what messages the users sign. However, if a signer is forced to either acknowledge or deny a signature, e.g., in court, she cannot falsely deny her real signatures. There are even *unconditionally secure* signature schemes, where even an adversary with unlimited computing power cannot produce forgeries better than by using a random number and hoping that it is the signature [CR91, PW92]. However, unconditionally secure signatures seem to be impractical, in contrast to fail-stop and undeniable ones.

All these new schemes exist in several *variants* with different properties. For instance, fail-stop signatures can be more efficient if all signatures of one signer have the same recipient (which is possible in applications like payment systems). Another example is that the original unconditionally secure schemes do not enable recipients to transfer signatures from one to another (which one tacitly takes for granted with most other digital signature schemes), but one can reintroduce this property in a limited way. A third example is undeniable signatures where signers can unconditionally deny forged signatures, which is a slightly restricted fail-stop property [CHP92].

This situation alone might call for some classification. However, it would not be too bad if all these schemes were signature schemes in the sense of the GMR definition with some additional properties. However, none of them is, because they all differ somehow in their *structure*, i.e., in the number of algorithms and their parameters. For instance, in fail-stop signature schemes, the recipients (or a center trusted by them) must take part in key generation, so that they can be protected from the signer falsely proving her own signatures to be forgeries. With undeniable signatures, there is no test algorithm in the usual sense, but an interactive protocol, usually a zero-knowledge proof, where the signer proves the correctness of the signature to the recipient. Moreover, in both these types of schemes, a court cannot just test signatures brought by a recipient, but must ask the signer, too (to give her a chance to produce a proof of forgery, or to carry out a verification or disavowal protocol, respectively). With unconditionally secure signatures, there is no public key in the usual sense, but everybody obtains a different test key (although they are of course somehow related) in a complicated key generation protocol.

Finally, note that not even all signature schemes that are supposed to be ordinary, i.e., that have no special security properties, are covered by the GMR definition: Sometimes, signing is not just an algorithm that takes a secret key and a message (and at most the number of this message) as inputs, but some memory is needed between applications, e.g., Schnorr's scheme with preprocessing [S91]. In another example, there is a trusted public random value in addition to the public keys [BC92].

So what does one do with all these new types of schemes?

Even if they all had a new definition (which some do, but most variants don't), the situation would be unsatisfactory: First, if they are all called signature schemes (which nobody seems to challenge), they ought to have something in common. Secondly, new definitions in cryptology are just as error-prone as new systems; hence one should have as few definitions as possible and evaluate them carefully.

Hence the properties common to all signature schemes should be stated clearly once and for all. Special properties should be formulated as additions to this general definition. Applications then know what to expect from all signature schemes, or can list the additional properties they need. This is what we try to achieve in this paper.

2 The Main Ideas

Intuitively, what all signature schemes have in common is exactly what was written in the very first sentence: they provide message authentication such that disputes about the authenticity of messages can be settled by third parties. However, as sketched above, they have different *degrees of security*, such as fail-stop, and their *structure* varies widely. All existing definitions specify what disputable message authentication means only in terms of one given structure and with one particular degree of security. This is why the whole definition had to be changed whenever the structure was modified or a different degree of security was introduced, although new definitions should basically represent the same service.

The basic idea to obtain a general definition is therefore to separate the definition of service, structure, and degree of security. Hence one has to define

- the minimal service (i.e., message authentication that is provable to third parties) in an ideal way, independent of the structure and the degree of security; this can be done in temporal logic here,
- a very general structure common to all signature schemes, and then
- what it means that any system provides any service with a certain degree of security.

Given such a general definition, one can classify particular types of signature schemes according to

- additional service properties,
- special (simpler) cases of the structure, and
- the actual degree of security.

An additional advantage of such a modular definition is that only the service is special to signature schemes, whereas the rest can be reused with other schemes that still lack general definitions, e.g., digital payment systems. More generally, some aspects of the degrees of security are the same for all types of reactive systems; and the structure and the remaining aspects of the degree of security characterize a class of schemes that might be called cryptologic transaction schemes.

The scope of signature schemes actually defined here is schemes that could be used in the place of hand-written signatures in law, i.e., where each signer has complete control over what she signs. This excludes blind and group signature schemes — their services are too different for a joint definition. The general parts of the definition and classification, however, could be applied to them, too.

3 Related Work

There are two somewhat related approaches at general and abstract definitions of cryptologic systems: definitions of secure multi-party computation protocols from theoretical cryptology and logics of authentication (BAN-logic) from the design of protocols using cryptologic primitives. The relations and differences to those approaches are now discussed.

3.1 Relation to Secure Multi-Party Computation Protocols

Deriving a cryptologic definition from an ideal service specification is new with signature schemes, but it is known with multi-party function computation. (Overviews can be found in [B91, MR91, FGY92].) However, existing definitions were not general enough to be used for signature schemes. The following new features were needed:

- *Logic specification.* The services of different types of signature schemes are similar, but not identical. In multi-party function computation, however, the specification is given by just one ideal trusted host, which has to be simulated very closely by the actual distributed cryptologic protocol. Here, a logic specification has to be used instead of the trusted-host specification. Hence one can formulate some minimal requirements that have to be fulfilled by all signature schemes, and add additional requirements to specify particular types of schemes. (Actually, we can show that not even individual types of signature schemes, e.g., the ordinary ones according to the GMR definition, simulate one ideal trusted host.)
- *Reactive systems.* Current formal definitions of multi-party protocols only treat the computation of functions. Each user of such a system makes one initial input and obtains one output at the end. Signature schemes, however, are reactive systems: They can go on (polynomially) forever, and at any time users can decide what messages they want to sign or what disputes to carry out. (An informal sketch of more general games exists in [GMW87]; if they were formalized they would be nearer to signature schemes in this respect.)
- *Varying connections.* Usual multi-party protocols assume fixed connections throughout a protocol run, usually reliable point-to-point channels between each pair of participants and reliable broadcast channels. With signature schemes, the connections and their security can vary as a function of time: One cannot assume that the participants, which may be hand-held computers in pockets, have so many dedicated channels, and the channels are not necessarily secure — this is part of the reason to use signatures. (More formally, this means that entities cannot use identities as addresses in general.)

3.2 Relation to BAN-Logic

Logic specification of signature schemes may sound rather similar to the logics of authentication from [BAN89] and many successive papers. However, there are several differences in the purpose and consequently in the approach. This is natural, since these logics were primarily designed to find errors in key exchange protocols with symmetric cryptosystems, which they do successfully.

- Logics of authentication are used to specify and prove protocols *using* cryptologic primitives, whereas here, the signature schemes themselves are to be specified.
- Logics of authentication assume very simple cryptologic primitives; hence they deal with individual algorithms and keys of those schemes. Hence a new type of logic would be needed for each new type of signature scheme, just as with current cryptologic definitions. In [LABW92], e.g., integrity is still assumed to be provided by a variant of an encryption scheme, which is a structure that most signature schemes definitely do not have.
- If logics of authentication have a formal semantics, not the real cryptologic schemes are used, but algebraic abstractions [DY81, AT91]. Hence, if one finds an error with the logic, there was an error in the real protocol, but if one proves the algebraic abstraction to be

secure, one is not sure that the real protocol is secure, too. Here, the real signature schemes are the semantics.

In future, the approaches might meet in logics for the design of complicated protocols, but using more general primitives and a real cryptologic semantics.

4 Service Specification of Signature Schemes

As explained in Section 2, we want a logical specification of the service of all digital signature schemes, independent of the structural differences mentioned in Section 1.2. This can only be done by considering the whole signature scheme as one system whose behavior at the *interface* to its users is specified, see Figure 1. This is completely normal with service specification in the field of specification of distributed systems (see, e.g., [L83]), but unusual in cryptology.

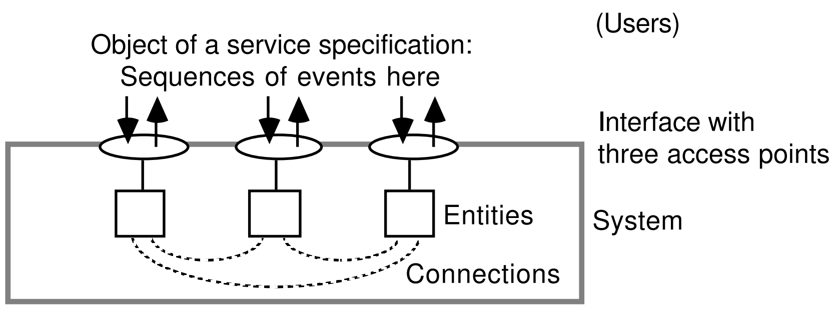

Figure 1 A system and its interface

The entities, in our case, correspond to processes running on the behalf of individual users. For instance, if the signature scheme is implemented on hand-held computers used by individuals, an entity corresponds more or less to the processes on one such computer. The users of the signature scheme are the environment, hence they are not explicitly modeled, but intuitively one can assume that each access point is owned by a particular user. (This becomes more important when degrees of security are modelled below and users are considered as mutually distrusting.)

4.1 Interface Events

So what are the interface events in a signature scheme, i.e., what are the users concerned with, not just the entities inside the system? Informally, they need the system to perform three types of so-called transactions, each consisting of a few related interface events at several access points.

- **Authentication.** Any two users, called a signer and a recipient, have to be able to authenticate a message. The signer inputs which *message* is authenticated. The recipient has to know the message, too, and the *identity* of the signer who is supposedly authenticating it. At the end, the system must give the recipient a Boolean output indicating whether the message was *accepted* as authenticated or not.

- **Disputes.** In a dispute, the third party trying to settle it and the recipient who claims to have the authenticated message take part, and sometimes the supposed signer, too. Again, the users have to input the *message* that is disputed and the *identity* of the signer who is supposed to have authenticated it, and the third party must obtain a Boolean output indicating whether the message was *accepted* as authenticated or not.
- **Initialization.** For technical reasons, initialization is needed. It corresponds to key generation and distribution inside the system, but the users only need to know the *identity* of the (future) signer for whom this is happening and whether initialization is *successful*. All users should take part in initialization.

Hence all the interface events only deal with messages, identities, and Boolean values denoting whether something was accepted or not, not with signatures and keys. More formally and more precisely, the domain of the interface events are defined as follows: Each event is presented as a triple

$$(identity, direction, value),$$

where *identity* denotes the access point where the event occurs, $direction \in \{in, out\}$ denotes whether it is an input or an output event, and *value* is the value that is passed through the access point. The domain of *identity* is a set $Id = Id_S \cup Id_R \cup Id_T$, i.e., separate access points are used for acting as signer, recipient, or third party.

The domains of *value* are a bit more complicated than sketched above. They are shown in Table 1. The individual parameters in the values have the following meaning:

- Each input value starts with an identifier, such as *"sign"*, that characterizes the type of action wanted.
- Parameters m mean a message from a message space M.
- Parameters id_S mean an identity of a signer, i.e. an element of Id_S.
- Parameters ids_R are descriptions of sets of recipients. Their domains may be arbitrary subsets of the power set of Id_R. They are needed because inside some signature schemes, signing depends on the identities of the intended recipients. The empty string ε is used as the description of the complete set, i.e., for "no restriction" — this is the standard case.
- N is a message bound, i.e., the number of messages the signer can authenticate at this access point after this initialization.
- The two values *acc* in bold face denote the acceptance of a message as authenticated. Disputes in some schemes may have results other than Boolean, hence an arbitrary domain $Dispute\_Results \supseteq \{\text{TRUE}, \text{FALSE}\}$ is allowed. The two most important cases are
  - acc = *"not_cooperated"* in undeniable signature schemes means that the signer refuses to cooperate and no decision can be reached in the system. (The users outside the system must decide what to do with non-cooperating signers under which circumstances.)

- *acc* = *“broken”* in fail-stop signature schemes means that a valid proof of forgery has been shown in the system. This must be distinguished from the result that the signature has simply not been accepted, because one may wish to stop the system.

- The parameters ids_{in} in initialization denote which access points (and thus users) are supposed to take part, and ids_{out} denotes which of them actually took part successfully. (This is needed to cover schemes with interactive key generation where disrupters have to be excluded.) *acc* denotes whether initialization was successful at all, i.e., whether it is now possible to authenticate messages under id_S.
- The outputs *eot* in some transactions are usually just needed to denote that the transaction has ended.

		Values		
		Signer	Recipient(s)	Third party/ies
Authen-tication	In:	***“sign”, m***, ids_R	***“test”, m***, $\boldsymbol{id_S}$	—
	Out:	*eot*	***acc*** $\in$ {TRUE, FALSE}	—
Dispute	In:	*“disavow”*, *m*, ids_R	***“show”, m***, $\boldsymbol{id_S}$	***“decide”, m***, $\boldsymbol{id_S}$, ids_R
	Out:	*eot*	*eot*	***acc*** $\in$ *Dispute_Results*
Initial-ization	In:	***“init”***, ids_{in}, N	***“init”***, $\boldsymbol{id_S}$, ids_{in}, N	***“init”***, $\boldsymbol{id_S}$, ids_{in}, N
	Out:	*acc*, ids_{out}	*acc*, ids_{out}	*acc*, ids_{out}

Table 1 Types of interface events, sorted according to the access points where they occur and the transactions to which they belong. The most important ones are in bold face; they are the ones that would also be needed if just ordinary digital signature schemes were specified.

Note that the signer does not input her own identity during authentication — it is implicit in the access point she uses (usually in the form of a secret key stored in the entity serving the access point). How the access to the access point (and thus the usage of the key) is restricted is not the business of the signature scheme. Moreover, a service where one could input any identity at any access point would be insecure: One could just input the identity of someone else. Similarly, the signer does not input her identity in initialization (in that case, it is present in the form of access to a broadcast channel under this identity, so that the generated key is linked to the identity).

4.2 Some Remarks on Generality

It seems natural at this point (judging from the reaction of some readers of preliminary versions) to come up with many suggestions for more transactions and interface events. For instance, why can the signer not agree to a message in a dispute, instead of disavowing it? Or couldn't one authenticate a message for several recipients at once? What about using several third parties, and the majority prevails?

However, a *general* definition has to contain *few and restricted* transactions.

The reason is that all transactions required in the minimal specification have to be offered by *all* signature schemes, i.e., they have to produce sensible reactions (to be logically specified below) on all the input events from the minimal domains specified here. If one wants more transactions or types of input events, they are additional service properties that some schemes offer and others don't, and can serve for a classification.

Conversely, why have the sets of identities of recipients (which make the logic requirements rather complicated) then been added? The reason is that they are *permitted restrictions*, not additions: The domains were not the power set of Id_R, but arbitrary subsets. Hence one can, e.g., specify signature schemes where $|ids_R| = 1$ is required, i.e., only signatures specific to certain recipients can be produced, not general ones. One can even allow that additional predicates are given that restrict some of these parameters as functions of others, e.g., a predicate $suitable(Id_R, ids_{out}, ids_R)$ that describes which sets of identities, ids_R, are suitable inputs by the third party in a dispute if the set Id_R of all identities and the set ids_{out} of successful participants in initialization are given.

Another generalization by restriction that we can make is including accountable centers, e.g., key distribution centers that can be held responsible if they cheat. (Schemes with centers that must be trusted would not seem to deserve the name digital signature schemes.) However, in a short presentation like this they are better omitted.

4.3 Specification and System Parameters

The domains of interface events depends on some basic domains, such as the message space M. Hence there is in fact a whole family of specifications $Spec_v$, where the parameter v denotes a tuple of these basic domains. These domains will be treated in two different ways:

- Almost all parameters can be fixed by the designer of an individual signature scheme. That is, there may be signature schemes for special message spaces or with special sets *Dispute_Results* etc. This makes the definition very general.
- The exception is that any signature scheme will have to work for arbitrary sets Id_S, Id_R, and Id_T of identities; see Section 5.

The former type of parameters are called *specification parameters*, the latter *system parameters*.

4.4 The Logic Requirements

The logic requirements on the signature schemes specify what outputs the system has to produce, depending on previous interface events. Hence they are formulas that describe some sequences of interface events as permitted and others as forbidden. Recall that these are ideal requirements — error probabilities etc. are handled once and for all in the degrees of security.

Note that we are lucky that all minimal requirements on signature schemes just specify what outputs may occur: There are neither privacy requirements nor probability distributions. (Of course, there will be secret information inside the systems, but this is just because no other way of realizing the service is known.) Hence normal temporal logic can be used for the specification.

Each requirement is formulated from the point of view of some *interested parties*, formally a set of identities. It only involves events at the access points of these parties. This is needed because in Section 6 a requirement has to be fulfilled even if all the other parties are attacking and their entities deviate from the protocol.

4.4.1 Overview

First an informal overview of the requirements is given.

- **Requirement of the recipient on disputes.** Once a recipient has accepted a certain message m as authenticated by a certain signer id_S, i.e., he made an input ($\text{“}test\text{”}$, m, id_S) and obtained the output acc = TRUE, the message should also be accepted by a third party in a dispute (i.e., the output acc of the third party in that dispute should be TRUE). To include schemes like undeniable signature schemes, it has to be tolerated that disputes only work if the signer cooperates.

  The interested parties are the recipient and the third party who settles the dispute. Note that the whole requirement is formulated from their point of view: no event at the access point of the signer is mentioned. (Informally, only the recipient might be called “interested”, but the requirement presupposes honesty of the third party and can only be fulfilled if the third party’s entity is correct.)

- **Requirement of the signer on disputes.** As long as a signer has not authenticated a certain message, a third party should not accept that she did, at least if she cooperates in disputes. The interested parties are the signer and the third party.

- **Effectiveness of authentication.** If a signer and a recipient want to carry out authentication (for the same message and identities), the message should be accepted, under the precondition that correct initialization took place (i.e., it ended with acc = TRUE).

- **Effectiveness and correctness of initialization.** If an initialization is started, and at least the future signer and one further participant are interested, it should end successfully. Moreover, the outputs denoting whether it worked at all, and with whom, should be consistent among any set of interested parties, and nobody who took part correctly should be missing from the set of successful participants.

- **Availability of service.** Authentication, disputes, and initialization should always end within reasonable time, so that new inputs are accepted at the same access point.

One may miss **unforgeability** in this list, i.e., if a signer has not authenticated a certain message, a recipient should not accept that she did. However, one can show that unforgeability is a consequence of the first two requirements (with one exception of only technical interest). Moreover, there are applications where unforgeability is not required for its own sake.

In the following, some details about the type of formalization and one formalized requirement are presented.

4.4.2 Some Details

The first extension is that varying *connections* (see Section 3.1) have to be handled. For instance, for the requirement of the recipient on disputes to be fulfilled, the entities of the recipient and the third party must be connected for the time of the dispute (e.g., to forward a signature). However, no preconditions about the connections during authentication are needed. Hence the necessity of connections has to be specified in the requirements. For this, a variable *connections* is introduced. It contains, for each round, the connections that are in place between this round and the next, in terms of identities and channel types, such as "point-to-point" or "broadcast". Only point-to-point channels are allowed in authentication and disputes, whereas broadcast channels can be used in initialization.

Secondly, a model of *time* is needed. We consider a simple synchronous model. In each round, inputs can be made at any access points, and then outputs at some access points occur. This may have to be extended in future.

Thirdly, it is assumed for simplicity that participants who carry out a *transaction* together start it synchronously. This is no great loss of generality, since they must be synchronized, at least moderately, and this is not really a task of the signature scheme. Moreover, just one transaction is permitted at a certain access point at a given time (whereas transactions at different access points may overlap), so that the entities of the signature scheme need not bother with scheduling.

Finally, one has to decide to what extent the requirements contain preconditions, i.e., to what extent the honest users must *"behave reasonably"* in order to have any guarantees. For instance, one has to decide whether the requirements have preconditions that initialization has been performed before any other transaction. Minimal requirements for a general definition are obtained by strong preconditions. More user-friendliness (e.g., error messages in such cases), can be added as additional service properties.

4.4.3 One Formalized Requirement

The requirement of the recipient on disputes is now presented formally. Note, however, that this is already almost half the formalization of all requirements, since many following predicates (those about transactions, connections, and initialization) are not specific to this requirement.

The operators from temporal logic are written out, like SINCE, so that readers unfamiliar with temporal logic can guess what is meant. For definitions, however, see [MP91]. (The operator PREVIOUS, which denotes that something happened in the previous round, is defined here to evaluate to TRUE in the first round. NEXT in the last round is treated analogously.)

Basic Predicates About Transactions

The set of events in each round is denoted by a variable *events*. First some basic predicates are introduced. Let

$input\_at(id) := \exists value'\colon (id, in, value') \in events;$

$output\_at(id) := \exists value'\colon (id, out, value') \in events.$

An input is defined as valid if no other transaction is currently executed at the same access point, i.e., no other input has been made there since the last output (or since the system start). This can be written

$$valid\_input(id, value) := (id, in, value) \in events \wedge \text{PREVIOUS}\,\big((\neg input\_at(id))\ \text{WEAK\_SINCE}\ output\_at(id)\big).$$

Similarly, if an output is considered, the previous valid input at the same access point (i.e., the user input on which the system is currently reacting) is characterized by

$$previous\_valid\_input(id, value) := \text{PREVIOUS}\,\big((\neg output\_at(id))\ \text{SINCE}\ valid\_input(id, value)\big).$$

The last predicate in this group means that no output occurs at a set of access points, usually an interest group:

$$no\_output\_at(interest\_group) := \forall id \in interest\_group\colon \neg output\_at(id).$$

Simple Predicates About Accepting and Disputes

Now one can define what it means that a recipient id_R accepts a message m as authenticated by a signer id_S:

$$recipient\_accepts(id_R, m, id_S) := (id_R, out, \text{TRUE}) \in events \wedge previous\_valid\_input(id_R, (\text{'}test\text{'}, m, id_S)).$$

The next predicate denotes that a recipient id_R and a third party id_T start a dispute about the message m consistently (and with a certain parameter ids_R):

$$shows\_to(id_R, id_T, m, id_S, ids_R) := valid\_input(id_R, (\text{'}show\text{'}, m, id_S)) \wedge valid\_input(id_T, (\text{'}decide\text{'}, m, id_S, ids_R)).$$

The desired results of such a dispute are modeled next; the word “weakly” stands for the possibility of the output *‘not_cooperated’*.

$$recipient\_weakly\_wins(id_R, id_T) := \big((id_T, out, \text{TRUE}) \in events \vee (id_T, out, \text{'}not\_cooperated\text{'}) \in events\big) \wedge \exists eot\colon (id_R, out, eot) \in events.$$

Connections

Before the previous predicates can be put together in the requirement, formalizations of correct connections and (seemingly) correct initialization are needed.

The values of the variable *connections* are sets of connection wishes of the form (*transaction_type*, *list*), where *list* is a list of identities and *transaction_type* $\in$ {*‘authentication’*, *‘dispute’*, *‘init’*}. Above, it was sketched that the input would specify the channel type, but the current version is more general: One can use structures with different channel types for the same transaction types. However, with signature schemes, this is only relevant for initialization. The list must fit the transaction type; e.g., it must consist of one signer and one recipient in authentication, and of a recipient, a third party, and possibly a signer in disputes.

We assume two functions *connection_types* and *type_of*. The former maps each *transaction_type* to appropriate list types, and the latter maps lists to their types.

Now a correct connection during a transaction is defined. (With signature schemes, the following complicated formalization is mainly needed because of initialization.) Requirements are considered from the point of view of an interest group and thus from a subset, *subset*, of the intended participants of the transaction (those members of the interest group that intend to take part). Hence there should be an input (*transaction_type*, *list*) where the list comprises this subset and is of the correct type. Moreover, the subset should be connected in the same way in each round of a transaction, i.e., the identities must appear in the same positions in the lists throughout the transaction. Let *positions_in*(*subset*, *list*) denote those positions (i.e., a mapping from *subset* to $\mathbb{N}$). Then, for a variable *pos* which denotes such positions, let

$$\begin{aligned}&\mathit{correct\_connection}(\mathit{transaction\_type}, \mathit{subset}, \mathit{pos}) := \exists \mathit{list}:\\ &\quad (\mathit{transaction\_type}, \mathit{list}) \in \mathit{connections}\\ &\quad \wedge \mathit{positions\_in}(\mathit{subset}, \mathit{list}) = \mathit{pos}\\ &\quad \wedge \mathit{type\_of}(\mathit{list}) \in \mathit{connection\_types}(\mathit{transaction\_type}).\end{aligned}$$

Initialization

The interest group in the requirement of the recipient on disputes is a set $\{id_R, id_T\}$ with $id_R \in Id_R$ and $id_T \in Id_T$. Hence we can concentrate on initialization from their point of view.

The first predicate means that neither the recipient nor the third party has tried an initialization for id_S before the current round:

$$\begin{aligned}&\mathit{no\_init}(\{id_R, id_T\}, id_S) :=\\ &\quad \forall id \in \{id_R, id_T\}\ \forall ids_{in}\ \forall N:\\ &\quad \text{PREVIOUS ALWAYS\_IN\_THE\_PAST} \neg\, \mathit{valid\_input}(id, (\text{‘}\mathit{init}\text{’}, id_S, ids_{in}, N)).\end{aligned}$$

If one initialization for id_S has already taken place, the fact that the recipient or the third party tries a second initialization for id_S can be expressed by

$$\begin{aligned}&\mathit{second\_init}(\{id_R, id_T\}, id_S) :=\\ &\quad \exists id \in \{id_R, id_T\}\ \exists ids_{in}\ \exists N: \mathit{valid\_input}(id, (\text{‘}\mathit{init}\text{’}, id_S, ids_{in}, N)).\end{aligned}$$

The next predicate means that the recipient and the third party give consistent inputs to start initialization for id_S, and that id_R is an intended recipient according to this initialization.

$$\begin{aligned}&\mathit{correct\_init\_start}(\{id_R, id_T\}, id_S, ids_{in}, N) :=\\ &\quad (\forall id \in \{id_R, id_T\}: \mathit{valid\_input}(id, (\text{‘}\mathit{init}\text{’}, id_S, ids_{in}, N)))\ \wedge\ id_R \subseteq ids_{in}.\end{aligned}$$

Similarly, the next predicate models that an initialization comes to a consistent end in the current round with the output ids_{out}.

$$\begin{aligned}&\mathit{correct\_init\_end}(\{id_R, id_T\}, ids_{out}) :=\\ &\quad \forall id \in \{id_R, id_T\}: (id, \mathit{out}, (\text{TRUE}, ids_{out})) \in \mathit{events}.\end{aligned}$$

A predicate that is true in a round where a first correct initialization ends is presented next. One condition is that there has been a correct start of an initialization. Since this start, no other outputs must have occurred and correct connections must have been provided. However, one cannot use the operator SINCE for this immediately, because the connections must already be in place in the round of the start, and they are no longer needed in the round with the end. The same holds for the absence of other outputs. A modified operator is introduced for this situation: For arbitrary formulas a and b, let

$$a \text{ MODIFIED\_SINCE } b \;:=\; b \;\vee\; \text{PREVIOUS}\,(a \;\text{ SINCE }\; (a \wedge b)).$$

Moreover, let $Ids_{in}(Id_R)$ be the domain of the parameter ids_{in} as a function of Id_R Thus the predicate for correct initialization looks as follows.

$$\begin{aligned}
&correct_initialization(\{id_R, id_T\}, id_S, ids_{out}, N) := \\
&\quad correct_init_end(\{id_R, id_T\}, ids_{out}) \\
&\quad \wedge \exists pos\ \exists\ ids_{in} \in Ids_{in}(Id_R): \\
&\qquad\qquad ((no_output_at(\{id_R, id_T\}) \\
&\qquad\qquad\quad \wedge\ correct_connection(\text{'}init\text{'}, \{id_R, id_T\}, pos)) \\
&\qquad\qquad \text{MODIFIED_SINCE } correct_init_start(\{id_R, id_T\}, id_S, ids_{in}, N)).
\end{aligned}$$

Finally, it is represented that an interest group is in a correctly initialized state at the beginning of the current round, i.e., it has performed one correct initialization and tried no further ones before the current round.

$$\begin{aligned}
&correct_init_use(\{id_R, id_T\}, id_S, ids_{out}, N)) := \\
&\quad \text{PREVIOUS } (\neg second_init(\{id_R, id_T\}, id_S) \\
&\qquad\qquad\qquad \text{SINCE } correct_initialization(\{id_R, id_T\}, id_S, ids_{out}, N)).
\end{aligned}$$

Disputes Including Connections

The correct connections in a dispute from the point of view of a recipient and a third party can be represented more simply than above:

$$\begin{aligned}
&properly_connected(\text{'}dispute\text{'}, \{id_R, id_T\}) := \\
&\quad \exists list: (\text{'}dispute\text{'}, list) \in connections \\
&\qquad\qquad \wedge (list = (id_R, id_T) \vee\ \exists id_S \in Id_S: list = (id_R, id_T, id_S)).
\end{aligned}$$

The following predicate means that *if* a dispute is started correctly in the current round and correct connections are provided, it yields the correct result. The timing is as follows: There should be no other output until the correct result, with the exception that it is no longer specified what the system does if the connections were wrong in the previous round. There is an exception to the exception, however (unless one excludes that the result occurs in the very first round): In the first round, the connections of the previous round cannot be considered.

$$\begin{aligned}
&recipient_can_weakly_convince(id_R, id_T, m, id_S, ids_R) := \\
&\quad shows_to(id_R, id_T, m, id_S, ids_R) \\
&\quad \rightarrow (recipient_weakly_wins(id_R, id_T) \\
&\quad\ \ \vee no_output_at(\{id_R, id_T\}) \\
&\quad\quad \wedge \text{NEXT } (no_output_at(\{id_R, id_T\}) \\
&\qquad\qquad\qquad \text{UNLESS } (recipient_weakly_wins(id_R, id_T) \\
&\qquad\qquad\qquad\qquad\quad \vee \text{PREVIOUS } \neg properly_connected(\text{'}dispute\text{'}, \{id_R, id_T\})))).
\end{aligned}$$

Finally, recall the predicate $suitable(Id_R, ids_{out}, ids_R)$ that restricted sets of identities (Section 4.2).

The Requirement

Now the requirement (with the interest group as formal parameters) has the following precise meaning: If at any time the recipient accepts and correct initialization had already been carried out, then from the next round onwards, the recipient wins disputes (if the third party uses a suitable parameter ids_R), unless the recipient or the third party try a new initialization for the same signer's identity id_S.

$$
\begin{aligned}
&Req_{rec}(id_R, id_T) := \\
&\quad \forall id_S \in Id_S\ \forall m \in M\ \forall ids_{out} \in \mathcal{P}(Id)\ \forall N \in Message\_bounds\text{:} \\
&\quad \text{HENCEFORTH}\ \big((recipient\_accepts(id_R, m, id_S) \\
&\qquad\qquad\qquad\qquad \wedge correct\_init\_use(\{id_R, id_T\}, id_S, ids_{out}, N)) \\
&\qquad\qquad\qquad\qquad \rightarrow \text{NEXT}\ \forall ids_R\text{: } ((suitable(Id_R, ids_{out}, ids_R) \wedge id_R \in ids_R) \rightarrow \\
&\qquad\qquad\qquad\qquad\qquad (recipient\_can\_weakly\_convince(id_R, id_T, m, id_S, ids_R) \\
&\qquad\qquad\qquad\qquad\qquad \text{UNLESS}\ second\_init(\{id_R, id_T\}, id_S)))\big).
\end{aligned}
$$

The second initialization with the same signer has to be excluded because inside the system, it would lead to inconsistent keys, e.g., the third party's key might no longer fit the old signature. (Of course, one can still allow new keys for the same *user* by including a time stamp in the identity.)

4.5 Additional Service Properties for a Classification

Additional service properties that some signature schemes have can be sorted as follows.

Stronger requirements, i.e., concretizations where the minimal specification is ambiguous. The following types exist in actual schemes:

- *Strong requirement of the recipient on disputes.* The recipient wishes that any message he has accepted should also be accepted by a third party in a subsequent dispute. A weaker minimal requirement was made because of "undeniable" signature schemes. Most signature schemes, however, fulfil the recipient's original wish. This can easily be formalized by omitting the term "(id_T, *out*, *'not_cooperated'*) $\in$ *events*" in the definition of the predicate *recipient_weakly_wins*.
- *Strong requirement of the signer on disputes.* The minimal requirement of the signer only guaranteed her anything if she took part in disputes. The corresponding strong requirement omits this precondition. Then one need not find the signer to carry out a fair dispute.
- *Medium requirement of the signer on disputes.* An intermediate form is that if the signer does not take part in a dispute, *acc* may be FALSE or *'not_cooperated'*, but not TRUE, i.e., the message is not simply accepted. This is what "undeniable" signature schemes offer.
- *User-friendliness*, i.e., additional requirements on the reaction of the system on inputs that can be considered as user errors.

Fall-back specifications. Since specifications can be fulfilled with different degrees of security, it can make sense to have more than one specification. In particular, one can make strong requirements that will be guaranteed with a low degree of security (e.g., computationally), and weaker requirements that can be guaranteed with a higher degree of security, i.e., under more circumstances.

The *fail-stop* property is the only actual example of fall-back specifications. Informally, one can say that one first requires that no forgery ever occurs, but then, if a forgery occurs nevertheless, it should at least be provable. In the specification, this means that fall-back versions of both requirements on disputes are made: In the fall-back version, both the signer and the recipient tolerate that the third party's output in a dispute may be *"broken"* (or what they originally required). An additional requirement for the low degree of security is that the output *"broken"* never occurs (i.e., the system is not falsely stopped). The interest group is just the third party under consideration, i.e., the requirement holds even if the signer and the recipient collude.

Additional transactions (i.e., extensions of the domains of interface events) and corresponding requirements. Some existing ones are:

- A transaction *transfer* for a recipient to pass an authenticated message to a second recipient so that the second recipient is again sure that he can win disputes. This can work over an arbitrary chain of recipients, or just a fixed number of times. (Most digital signatures allow arbitrary transfers, but undeniable ones none at all, and different versions of unconditionally secure signatures allow no transfers or a fixed number.)
- If there is a fail-stop property, one usually also has a transaction to transfer the knowledge that the system has been broken.
- There can be more versions of initialization, e.g., one for new recipients to enter the system later. This is easy with ordinary digital signature schemes, but not with unconditionally secure schemes where key generation is interactive.
- Local transactions, such as queries about the number of messages that have already been authenticated.
- External verifiability of transactions. This means that the transactions can be carried out with additional participants who just observe and check if any of the original participants cheats (e.g., when a court is watched by the public). Details can be formalized independent of the transaction type, but they are omitted here.
- Distributed variants of transactions, such as authentication for several recipients at once in a way that the honest ones among them either all accept or all reject. (An example for a non-trivial case was given in [PW92b].)

Special specification parameters are also a type of concretization where the minimal specification is ambiguous (see Section 4.3). Examples are

- special message spaces and
- special dependencies on recipients.

Privacy requirements. The invisibility of undeniable signatures is the only privacy requirement that has so far been considered with signature schemes. (Remember that blind signatures need a separate specification.) One might also define directedness of signatures to mean that the authentication of a message is valid only for the intended recipient.

5 Structure

Since the service of digital signature schemes has been defined at an interface, one can consider implementations with arbitrary structure, e.g., even centralized ones. However, only decentralized ones are defined to *be* signature schemes: Two people meeting in the desert (with their hand-held computers) should be able to exchange authenticated messages. More precisely, there must be one entity per access point, as in Figure 1, and only the entities under the access points concerned take part in a transaction.

5.1 General Structure

A signature **scheme** just consists of the programs for such entities, i.e., it consists of a triple of probabilistic polynomial-time interactive algorithms,

$$Programs = (signer\_program, recipient\_program, third\_party\_program).$$

and the tuple of specification parameters (see Section 4.3). The programs must distinguish in- and outputs at the user interface and communication with other entities.

If one assumes that the variable *connections* only specifies transaction types, not channel types (see Section 4.4.3), the scheme must also contain the mapping from transaction types to channel types that this network entity performs in each round, i.e., a rudimentary program for this entity.

A **system** is an instantiation of a scheme with certain system parameters. These are the sets of identities (Id_S, Id_R, Id_T), as mentioned in Section 4.3, and additional security parameters. Given a scheme and the system parameters, the instantiation is defined as follows: There is one entity per identity, each running the appropriate program. The entities have all system parameters and their own identity as an initial state. The varying connections are handled by an additional entity that models the network in a very simple way: It has one access point where the desired connections can be input (this input is modeled by the variable *connections* from Section 4.4), and it switches the desired connections.

With any precise notion of an interactive algorithm, it is clear how global runs of such a system (in interaction with users) are defined.

Note that all structural details are nicely hidden under the interface and in the general structure: Within a transaction, the entities can carry out an arbitrary number of algorithms or interactive subprotocols, and they can use an arbitrary number of local variables from their memory.

5.2 Simpler Structures

Special structural properties are restrictions on the complexity of the general structure.

The most important one is non-interactive authentication, i.e., signature schemes where signatures in the normal sense exist. Generally, one can restrict any type of complexity (the number of rounds, the number of bits exchanged, the storage used), and for each transaction type separately.

Some special uses of memory (e.g., that certain secret information is read-only), which are of no particular interest as far as complexity is concerned, can be seen as the existence of secret and public keys in the sense of previous definitions.

6 Security

Security means that the systems derived from a scheme only produce sequences of interface events permitted by the service specification, even in the presence of certain attackers and possibly with small error probabilities.

6.1 General Security Definitions

Access of attackers to parts inside the system. When a requirement in the interest of a certain set of parties, like one recipient and one third party, is considered, only the entities of those parties are assumed to be correct, i.e., to run the programs prescribed in the scheme. Additionally, the channels between them are secure (except that normal point-to-point channels are only reliable, not private) and the entity modeling the connections is correct — recall that wrong or unreliable connections are modeled in the input *connections* to this entity.

Influence of attackers on honest users. Sequences of interface events are produced by an interaction of the system with arbitrary users. Note that the users are not specified, hence one must somehow quantify over their possible behaviors. Moreover, even the honest users from the interest group may be influenced by the attacker, e.g., in the choice of messages they authenticate. This corresponds to the active attacks of [GMR88]. In more general signature schemes, active attacks on recipients and third parties are also possible and dangerous (because all entities may have secret information — this was originally overlooked with unconditionally secure schemes). The most general active attack on any reactive system is shown in Figure 2. In the more intuitive Version 1, there is a universal quantifier over honest users who are more or less influenced by the attacker. (Such a general representation should hopefully make many further discussions unnecessary about which active attacks have to be considered in what types of systems, not just with signature schemes.) For requirements like ours, which only specify correct outputs, one can easily show that the simpler Version 2, where the attacker controls the honest users completely, is equivalent. Version 1 is needed for privacy requirements.

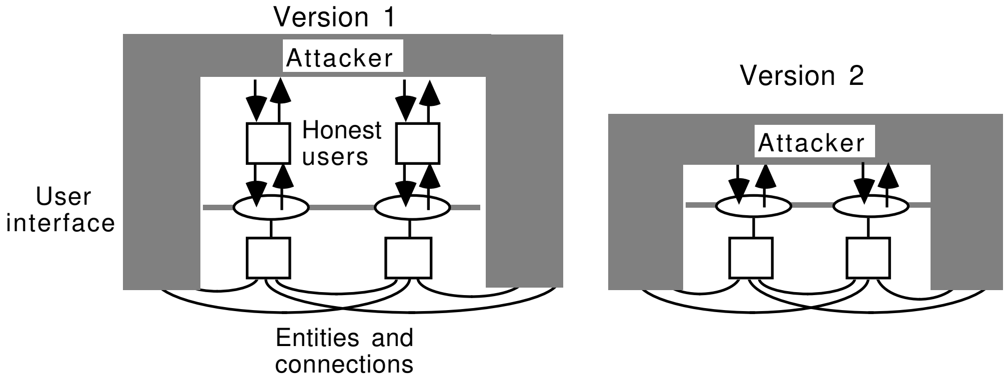

Figure 2 General active attack on two correct entities in a reactive system

Error probabilities and computational aspects. A general definition of the notion that a reactive system fulfils a logic requirement in the information-theoretic sense with and without error probabilities is now fairly straightforward. Similarly, computational security is defined by restricting the attacker strategy to polynomial-time computations. Note that the probabilities are taken over all random choices of the correct entities and the attacker strategy. The definitions of "small" error probabilities are in terms of the security parameters, which were represented in the initial state of the entities. If the notion of "small" is invariant against addition, then with two requirements, the conjunction of them is also fulfilled. Then the logic is sound in the resulting cryptologic semantics.

6.2 Combinations of Degrees of Security

Each requirement on digital signature schemes can be fulfilled with a different degree of security. One of the most important classification criteria for signature schemes is the degrees of security of the two requirements on disputes. The following combinations exist (where dual security is a new term):

- **Ordinary security** means that the requirement of the signer holds computationally and that of the recipient information-theoretically, as with ordinary digital signature schemes.
- **Dual security** is the other way round: The requirement of the signer holds information-theoretically and that of the recipient computationally, as in [CHP92]. (Note that it is now very easy to describe this degree of security and its difference to fail-stop security, which it wasn't before.)
- **Unconditional security** means that both requirements hold information-theoretically.
- **Fail-stop security** means that both original requirements hold computationally, but the fall-back service, as described above, is guaranteed information-theoretically. It is now easy to see that fail-stop security is stronger than both ordinary and dual security: If one identifies the output *"broken"* with FALSE (i.e., signatures are just rejected if the computational assumption has been broken), one obtains dual security, and if one identifies it with TRUE (i.e., signatures are still accepted even if the assumption has been broken), one obtains ordinary security.

Note, however, that one also has to consider whether the minimal or the strong requirements of the signer and the recipient are considered. In particular, one then sees that simple versions of fail-stop signature schemes only guarantee the weak requirement of the signer in both the original and the fall-back specification, but one can extend this so that the strong requirement of the signer is at least guaranteed computationally (i.e., if a signer cannot be found for a dispute, a computationally restricted attacker still cannot cheat her). One also has to mention which of the information-theoretic parts have error probabilities (always exponentially small).

7 Conclusion

We have shown that a general definition of signature schemes, covering all new types of them, can be given by

- requirements in temporal logic about the behavior of such schemes at a user interface,
- defining the structure in terms of entities of a distributed system, instead of the individual algorithms carried out in them, and by
- defining rather generally what it means that a reactive system fulfils a logic requirement in the presence of certain classes of attackers (both with access to some system parts and some influence on honest users) in the information-theoretic and computational sense.

Based on this definition, a systematic classification of additional properties of special types of signature schemes was also shown.

Acknowledgements

I am happy to thank *Johannes Buchmann* for first drawing my attention to the lack of a general definition of signature schemes, and *Joachim Biskup*, *Gerrit Bleumer*, *Andreas Pfitzmann*, and *Michael Waidner* for interesting and helpful discussions about this paper.

References

AT91 M. Abadi, M. Tuttle: A Semantics for a Logic of Authentication; 10th ACM Symposium on Principles of Distributed Computing (PoDC), acm press, New York 1991, 201-216.

B91 D. Beaver: Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority; Journal of Cryptology 4/2 (1991) 75-122.

BAN89 M. Burrows, M. Abadi, R. Needham: A Logic of Authentication; Operating Systems Review 23/5 (1989) 1-13.

BC92 J. Bos, D. Chaum: Provably Unforgeable Signatures; Crypto '92, Abstracts, Santa Barbara 1992, 1.1-1.8.

BCDP91 J. Boyar, D. Chaum, I. Damgård, T. Pedersen: Convertible Undeniable Signatures; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 189-205.

BPW91 G. Bleumer, B. Pfitzmann, M. Waidner: A remark on a signature scheme where forgery can be proved; Eurocrypt '90, LNCS 473, Springer-Verlag, Berlin 1991, 441-445.

C91 D. Chaum: Zero-knowledge undeniable signatures; Eurocrypt '90, LNCS 473, Springer-Verlag, Berlin 1991, 458-464.

CA90 D. Chaum, H. van Antwerpen: Undeniable signatures; Crypto '89, LNCS 435, Springer-Verlag, Heidelberg 1990, 212-216.

CHP92 D. Chaum, E. v. Heijst, B. Pfitzmann: Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 470-484.

CR91 D. Chaum, S. Roijakkers: Unconditionally Secure Digital Signatures; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 206-214.

D84 D. E. Denning: Digital Signatures with RSA and Other Public-Key Cryptosystems; CACM 27/4 (1984) 388-392.

DH76 W. Diffie, M. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644-654.

DSS92 The Digital Signature Standard Proposed by NIST; Communications of the ACM 35/7 (1992) 36-40.

DY81 D. Dolev, A. Yao: On the Security of Public Key Protocols; FOCS 1981, IEEE Computer Society, 1981, 350-357.

FGY92 M. Franklin, Z. Galil, M. Yung: An Overview of Secure Distributed Computing; Columbia University, Dep. of Computer Science, TR CUCS-008-92, March 24, 1992.

GMR88 S. Goldwasser, S. Micali, R. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. 17/2 (1988) 281-308.

GMW87 O. Goldreich, S. Micali, A. Wigderson: How to play any mental game - or - a completeness theorem for protocols with honest majority; STOC 1987, ACM, New York 1987, 218-229.

GQ91 L. C. Guillou, J.-J. Quisquater: Precautions taken against various potential attacks in ISO/IEC DIS 9796; Eurocrypt '90, LNCS 473, Springer-Verlag, Berlin 1991, 465-473.

HP92 E. van Heyst, T. Pedersen: How to make efficient fail-stop signatures; Eurocrypt '92, LNCS 658, Springer-Verlag, Berlin 1993, 366-377

HPP92 E. van Heijst, T. Pedersen, B. Pfitzmann: New Constructions of Fail-Stop Signatures and Lower Bounds; Crypto '92, Abstracts, Santa Barbara, August 1992, 1.9-1.14.

ISO91 ISO (International Organization for Standardization): Digital signature scheme giving message recovery; International Standard ISO/IEC 9796, Preliminary edition 1991-04-22.

L83 P. Linington: Fundamentals of the Layer Service Definitions and Protocol Specifications; Proceedings of the IEEE 71/12 (1983) 1341-1345.

LABW92 B. Lampson, M. Abadi, M. Burrows, E. Wobber: Authentication in Distributed Systems: Theory and Practice; ACM Transactions on Computer Systems 10/4 (1992) 265-310.

MP91 Z. Manna, A. Pnueli: The Temporal Logic of Reactive and Concurrent Systems – Specification –; Springer-Verlag, New York 1991

MR91 S. Micali, P. Rogaway: Secure Computation (Chapters 1-3); Laboratory for Computer Science, MIT, Cambridge, MA 02139, USA; distributed at Crypto '91.

P91 B. Pfitzmann: Fail-stop Signatures; Principles and Applications; Proc. Compsec '91, Elsevier, Oxford 1991, 125-134.

PW92 B. Pfitzmann, M. Waidner: Unconditional Byzantine Agreement for any Number of Faulty Processors; STACS 92, LNCS 577, Springer-Verlag, Heidelberg 1992, 339-350.

PW92b B. Pfitzmann, M. Waidner: Unconditionally Untraceable and Fault-tolerant Broadcast and Secret Ballot Election; Hildesheimer Informatik-Berichte 3/92, ISSN 0941-3014, Institut für Informatik, Universität Hildesheim, April 1992

RSA78 R. L. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of the ACM 21/2 (1978) 120-126, reprinted: 26/1 (1983) 96-99.

S91 C.P. Schnorr: Efficient Signature Generation by Smart Cards; Journal on Cryptology 4/3 (1991) 161-174.