1st ACM Conference on Computer and Communications Security, Fairfax, acm press 1993

Sorting Out Signature Schemes

Birgit Pfitzmann

Universität Hildesheim, Institut für Informatik,
Marienburger Platz 22, D-31141 Hildesheim, Germany
pfitzb@informatik.uni-hildesheim.de

Abstract. Digital signature schemes are a fundamental tool for secure distributed systems. It is important to have a formal notion of what a secure digital signature scheme is, 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, and although most signature schemes used in practice cannot be proved secure with respect to it, they are all built so that they hopefully fulfil it, e.g., by the inclusion of hash functions or redundancy to counter active attacks.

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, whose relation is not always clear. Obviously, 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, although there are similarities between the definitions. This is unsatisfactory.

This paper presents (an overview of) 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 cryptologic schemes.

Relations to secure multi-party protocols and 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 scheme 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. Moreover, 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 really takes 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 is 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 signer's help. (Hence they might rather be called "invisible" — not being deniable is common to all types of signatures.) This property is 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 signature schemes 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 signature schemes can be more efficient if all the 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 did 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 signature schemes 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. Undeniable signature schemes do not have a 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 a signature presented 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 disavowal protocol, respectively). In unconditionally secure signature schemes, there is no public key in the usual sense; instead, 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 the signature schemes that are supposed to be ordinary, i.e., that have no special security features, 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., in 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 schemes; 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 this paper tries to achieve.

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 is done in temporal logic here,
- a very general structure common to all signature schemes, and
- what it means that *any* scheme 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 schemes.

The scope of signature schemes actually defined here is schemes that could be used in the place of handwritten 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.

The service, both the minimal requirements used to define the notion of a signature scheme and additional ones, is discussed in Section 4. Sections 5 and 6 treat structure and degree of security. The parts that are common to much more general schemes than signature schemes can only be sketched here, and overviews of the properties special to signature schemes are given.

3 Related Work

There are two somewhat related approaches at general and abstract definitions of cryptologic schemes: 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. (Note that no knowledge of these approaches is assumed in the later sections.)

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, starting with [Y82]. (Overviews can be found in [B91, MR91, FGY92].) However, existing definitions were not general enough to be used for signature schemes. (This is no criticism — they go much further in other directions.) 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. Hence one can formulate some minimal requirements that have to be fulfilled by all signature schemes, and add further requirements to specify particular types of schemes. (Actually, one 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, yield 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 might 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.

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.

* Thanks to an anonymous reviewer, I noticed that very recently, [SM93] also formulated requirements independently of the structure and in temporal logic. (The differences in the schemes actually considered and the semantics remain.) The general idea of what should be expressed in requirements is indeed similar, except that [SM93] has no explicit interface. Introducing one might be helpful, because intuition seems to be needed to map, e.g., an accept statement from the requirements to something in the actual protocol, whereas here, it is simply the interface output in the protocol. I also found a separation of system entities and users helpful since entities have prescribed programs and users don't.

4 Service Specification of Signature Schemes

As explained in Section 2, we want a logic 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 regarding the whole signature scheme (more precisely: an instantiation of the scheme, see Section 5.1) as one system whose behavior at the *interface* to its users is specified, see Figure 1. This is entirely normal with service specification in the field of distributed systems (e.g., [L83]), but unusual in cryptology.

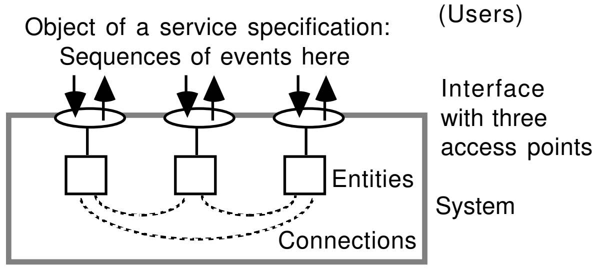

Figure 1 A system and its interface

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 modeled explicitly, but intuitively one can assume that each access point is owned by a particular user. (This becomes more important when degrees of security are modeled below and users are regarded as mutually distrusting.)

4.1 Domains of Interface Events

So what are the interface events of a signature scheme, i.e., what are the users concerned with, not just the entities inside the system? Informally, the users need the system to perform three types of so-called transactions, each consisting of a few related interface events at several access points. (Note that formally, this section, 4.1, only defines domains of interface events to be used in the requirements in Section 4.4. Showing how the events usually occur together in transactions only serves as motivation.)

- **Authentication.** Any two users, called a signer and a recipient, can 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. As in authentication, 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 obtains 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 the interface events only deal with messages, identities, and Boolean values denoting if something was accepted or not — *not* with signatures and keys.

More formally, 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 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 when acting as a 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 of the values have the following meaning:

- Each input value starts with a string, such as *'sign'*, that characterizes the type of action wanted.
- Parameters m denote a message from a message space M.
- Parameters id_S denote 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 if a message was accepted as authenticated. Disputes in some schemes may have results other than Boolean, hence an arbitrary domain $Dispute\_results \supseteq \{\text{TRUE}, \text{FALSE}\}$ is permitted. 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 message has simply not been accepted as authenticated, 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.

		Values		
		Signer	Recipient(s)	Third party/ies
Authentication	In:	***'sign'***, $\boldsymbol{m}$, ids_R	***'test'***, $\boldsymbol{m}, \boldsymbol{id_S}$	—
	Out:	*eot*	$\boldsymbol{acc} \in \{\text{TRUE}, \text{FALSE}\}$	—
Dispute	In:	*'disavow'*, m, ids_R	***'show'***, $\boldsymbol{m}, \boldsymbol{id_S}$	***'decide'***, $\boldsymbol{m}$, $\boldsymbol{id_S}$, ids_R
	Out:	*eot*	*eot*	$\boldsymbol{acc} \in \{\text{TRUE}, \text{FALSE}, \ldots\}$
Initialization	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 types of access points where they occur and the transactions they belong to. Those that would also be needed if only ordinary digital signature schemes were specified are in bold face.

- The outputs *eot* in some transactions are usually just needed to denote that the transaction has ended.

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 usually present in the entity in the form of access to a broadcast channel under this identity, so that the generated key can be distributed under the identity.)

4.2 Some Remarks on Generality

It seems natural at this point 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 specify *few and restricted* transactions.

The reason is that the 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 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, see Section 4.5.

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. One can even have additional predicates 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 Id_R is the set of all identities and ids_{out} that of successful participants in the corresponding initialization.

4.3 Specification and System Parameters

The domains of interface events had some parameters, such as a message space M. Hence there is a whole family of specifications $Spec_v$, where v denotes a tuple of these parameters. They will be treated in two different ways:

- **Specification parameters.** Almost all parameters can be fixed by the designer of a concrete signature scheme. This makes the definition very general. Hence there may be signature schemes for any message space M (say, with $M \subseteq \{0, 1\}^+$), any set $Bounds \subseteq \mathbb{N} \cup \{\infty\}$ of message bounds, any countable set $Dispute\_results \supseteq \{\text{TRUE}, \text{FALSE}\}$, any predicate *suitable*, and a few more parameters. The tuple of all specification parameters is called *spec_pars*.
- **System parameters.** 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.

4.4 Logic Requirements

4.4.1 Type of Requirements

The requirements on signature schemes specify what outputs the systems have 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 requirements are made in an ideal way — 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. Hence normal temporal logic (or other formalisms for describing sets of sequences) can be used. (Of course, there will be secret information inside the systems, but this is just because no other way of realizing the service is known. Not having to specify the secrecy of this information is a further advantage of having an interface and keeping keys below it.)

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.)

4.4.2 Overview of the Minimal Requirements

First, an informal and simplified overview of the requirements that all signature schemes must fulfil 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 (*'test'*, m, id_S) and the output was acc = TRUE, the message should also be accepted by a third party in a dispute, i.e., the third party's output acc should be TRUE. To include schemes like undeniable signature schemes, it must be tolerated that the output is *'not_cooperated'* instead.

  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 signer's access point is mentioned. (Informally, only the recipient might be called "interested", but the requirement presupposes that the third party is honest, and it 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 m, i.e., she has not input (*'sign'*, m, ids_R) for any ids_R, a third party should not accept that she did, i.e., its output acc in a dispute should be FALSE, at least if the signer disavows the message in this dispute. The interested parties are the signer and the third party.

- **Effectiveness of authentication.** If a signer and a recipient make consistent inputs to start authentication, i.e., with the same message and identities, the message should be accepted, i.e., the recipient should obtain the output 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, i.e., with acc = TRUE. Moreover, the outputs acc and ids_{out} 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 ids_{out}.

- **Availability of service.** Authentication, disputes, and initialization should 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 this is a consequence of the other requirements (with one exception of only technical interest). Moreover, applications exist where unforgeability is not required for its own sake.

The complete formalization of the requirements exceeds the scope of this paper. However, some general details and one example are presented.

4.4.3 Some Details

First, one has to decide to what extent one assumes honest users to *behave "reasonably"*. Minimal requirements for a general definition need strong preconditions. In particular, all the requirements have preconditions that initialization (with the correct interface events at the access points of the interested parties) has been performed before any other transaction for the same identity id_S of a signer. More user-friendliness (e.g., error messages in other cases), can be added as additional service properties.

Secondly, varying *connections* have to be handled (see Section 3.1). For instance, the requirement of the recipient on disputes can only be fulfilled if the entities of the recipient and the third party are connected during the dispute (e.g., to forward a signature), but 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 that contains, at each point in time, the current connections in terms of identities and channel types, such as "point-to-point" or "broadcast". Only point-to-point channels are permitted in authentication and disputes, whereas broadcast channels can be used in initialization. This variable can be visualized as one additional access point to the system where the connections can be controlled.

Thirdly, a model of *time* is needed. At present, a simple synchronous model is used. In each round, inputs can be made at any number of access points, and then outputs at some access points may occur. This may be extended in future.

Finally, it is assumed for simplicity that users who carry out a *transaction* together start it synchronously. This is without much 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 each 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.

4.4.4 An Example of the Formalization

The requirement of the recipient on disputes is now presented formally, but, for brevity, not from scratch in all details. (The rest of formalization should then be fairly clear, except for availability of service.) For definitions of temporal logic, see [MP91]. However, informal explanations are given for each formula and the operators from temporal logic are written with their names, such as SINCE, instead of their symbols, so that readers unfamiliar with temporal logic can guess what is meant.

The set of events at the user interface in each round is denoted by a variable *events*. First some basic predicates are introduced. 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 form of "since" is just what an operator BACK_TO expresses. Hence one can define

$$
\begin{aligned}
&valid\_input(id, value) := \\
&\quad (id, in, value) \in events \\
&\quad \wedge \text{WEAK\_PREVIOUS} \\
&\qquad ((\neg input\_at(id)) \text{ BACK\_TO } output\_at(id)),
\end{aligned}
$$

where

$$
\begin{aligned}
input\_at(id) &:= \exists value': (id, in, value') \in events; \\
output\_at(id) &:= \exists value': (id, out, value') \in events.
\end{aligned}
$$

(The operator WEAK_PREVIOUS, like PREVIOUS, denotes that something happened in the previous round, but it evaluates to TRUE in the very first round, in contrast to PREVIOUS.) 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

$$
\begin{aligned}
&previous\_valid\_input(id, value) := \\
&\quad valid\_input(id, value) \\
&\quad \vee \text{PREVIOUS} \\
&\qquad ((\neg output\_at(id)) \\
&\qquad \text{SINCE } (valid\_input(id, value) \wedge \neg output\_at(id))).
\end{aligned}
$$

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

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

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):

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

The desired results of such a dispute (in the interest of a recipient id_R and a third party id_T) are modeled next; the word "weakly" stands for the possibility of the output *'not_cooperated'*.

$$
\begin{aligned}
&recipient\_weakly\_wins(id_R, id_T) := \\
&\quad ((id_T, out, \text{TRUE}) \in events \\
&\quad \vee (id_T, out, \text{'}not\_cooperated\text{'}) \in events) \\
&\quad \wedge \exists eot: (id_R, out, eot) \in events.
\end{aligned}
$$

For lack of space, similar definitions of the following predicates are omitted:

- $recipient\_can\_weakly\_convince(id_R, id_T, m, id_S, ids_R)$, which means that if $shows\_to(\ldots)$ happens and the recipient and the third party are connected (according to the variable *connections*) until the next output, the next outputs will be $recipient\_weakly\_wins(\ldots)$,
- $correct\_init\_use(\{id_R, id_T\}, id_S, ids_{out}, N)$, which means that the recipient and the third party have already carried out initialization for id_S consistently and successfully (with the message bound N and the output ids_{out}), and
- $second\_init(\{id_R, id_T\}, id_S))$, which means that the recipient or the third party try a second initialization for the signer id_S.

Now the requirement of the recipient on disputes (with the interested parties as formal parameters) has the following precise meaning: If at any time the recipient accepts a message as authenticated from a certain signer and correct initialization had already been carried out, then from the next round onwards, the recipient will win any disputes occurring about that message (if the third party uses a suitable parameter ids_R according to the predicate $suitable(Id_R, ids_{out}, ids_R)$ from Section 4.2), unless the recipient or the third party try a new initialization for the same signer's identity id_S.

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

If one considers finite time, an operator WEAK_NEXT that evaluates to TRUE in the last round is needed instead of NEXT. The second initialization with the same signer has to be excluded because inside the system, it would lead to keys that do not fit the old signature. (Of course, one can still perform a new initialization for the same *user*, e.g., by adding a time stamp to the identity.)

4.5 Additional Service Properties

Additional service properties, which can be used to classify signature schemes, 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 is formalized by omitting the term "$\vee$ (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 guarantees her anything if she disavows the message. The corresponding strong requirement omits this precondition. In this case, 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., requirements on the reaction of the system on inputs that can be regarded as user errors.

One could also define *directedness of authentication* to mean that the authentication is only valid for the intended recipients.

Multiple 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 so called fall-back requirements that can be guaranteed with a higher degree of security, i.e., under more circumstances.

The *fail-stop* property is the only current example of multiple specifications. Informally, 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). It is made in the interest of just the third party in that dispute, i.e., even if the signer and the recipient collude.

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.

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 also sure that he can win disputes. This can work over arbitrary chains of recipients or just chains of fixed length. (Most digital signature schemes permit arbitrary transfers, but undeniable ones none at all, and different versions of unconditionally secure ones none or only over chains of fixed length.)
- 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.

One can also consider local transactions, such as queries about the status of an access points, and some distributed forms of the three minimal transactions, i.e., versions with more participants.

Privacy requirements. The invisibility of undeniable signatures is the only privacy requirement that has so far been considered with signature schemes. (Recall that blind signatures need a separate specification.)

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** consists of the programs for such entities and the specification parameters (see Section 4.3), i.e.,

$$Scheme = (spec\_pars, signer\_program, recipient\_program, third\_party\_program).$$

The "programs" are probabilistic polynomial-time interactive algorithms, which must distinguish in- and outputs at the user interface and on an arbitrary number of connections. (The usual formalization in cryptology is interactive Turing machines.) Note that these programs contain the real cryptologic algorithms, in contrast to system models used with logics of authentication.

A **system** is an instantiation of a scheme with certain system parameters *sys_pars*. These are the sets of identities (Id_S, Id_R, Id_T) (see Section 4.3) and additional security parameters. Given a scheme and the system parameters, the actual system is defined as follows: There is one entity per identity, each running the appropriate program. The entities have all the system parameters and their own identity as an initial state. The varying connections are handled by one additional entity that models the network in a simple way: It has one access point where the desired connections can be input (modeled by the variable *connections* in Section 4.4.3), and it switches these 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 the structural differences mentioned in Section 1.2 are nicely hidden under the interface: 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 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. Moreover, one can redefine notions such as public keys under certain structural restrictions.

6 Security

Security means that the systems derived from a scheme only produce sequences of interface events permitted by the service specification, but, in contrast to usual program validity of temporal logic, 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 certain parties, such as 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 from the scheme. Additionally, the channels between them are secure (except that normal point-to-point channels are only reliable, not secret) and the entity modeling the network is correct — recall that only connections that are explicitly required to be correct are modeled in the input *connections* to this entity.

With the minimal requirements on signature schemes, all attackers can be assumed to collude, i.e., all incorrect entities are replaced by one big attacker entity.

Influence of attackers on honest users. Sequences of interface events are produced by interaction of the system with arbitrary users. Even honest users who are interested parties 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 Figures 2 and 3. In the more intuitive version in Figure 2, there is a universal quantifier over honest users who are 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 cryptologic schemes.) For requirements that only specify correct outputs, one can easily show that the simpler version in Figure 3, where the attacker has direct access to the access points above the correct entities, is equivalent.

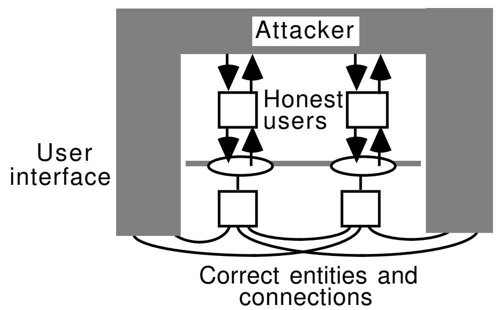

Figure 2 General active attack on two correct entities in a reactive system, version with indirect access

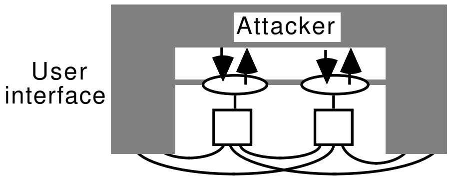

Figure 3 General active attack on two correct entities in a reactive system, version with direct access

Error probabilities and computational aspects. A general definition of the notion that a reactive scheme 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. There are only some difficulties with general dependencies of system parameters when more than one of them tends to infinity. Note that the probabilities are taken over all the 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, their conjunction is also fulfilled, so that the logic is sound in the resulting cryptologic semantics.

6.2 Combinations of Degrees of Security

Each requirement on a digital signature scheme can be fulfilled with a different degree of security. An important classification criterion 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 on disputes 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 on disputes 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 on disputes hold information-theoretically.
- **Fail-stop security** means that both original requirements on disputes hold computationally, but the fall-back requirements from Section 4.5 are fulfilled 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 on disputes are considered. In particular, one 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 version, but one can extend them so that the strong requirement of the signer is at least guaranteed computationally.

One also has to mention which of the information-theoretic parts have error probabilities (always exponentially small). It can be shown that error probabilities cannot be avoided with the requirement of the signer on disputes in dual and fail-stop security and with both requirements on disputes in unconditional security.

7 Conclusion

It has been 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 three programs for entities of a distributed system, instead of the individual algorithms carried out in them, and by
- defining rather generally what it means that a scheme 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 has been shown.

Parts of this approach have only been sketched, in particular the definitions of schemes and degrees of security (which are not special to signature schemes) and some theorems about general signature schemes. They can hopefully be presented in more detail in future work.

Acknowledgements

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

References

[AT91] M. Abadi, M. Tuttle: A Semantics for a Logic of Authentication; 10th PoDC, acm press, New York 1991, 201-216.

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

[BAN89] M. Burrows, M. Abadi, R. Needham: A Logic of Authentication; ACM 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, 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, Springer-Verlag, Berlin 1991, 441-445.

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

[CA90] D. Chaum, H. van Antwerpen: Undeniable signatures; Crypto '89, 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, Springer-Verlag, Berlin 1992, 470-484.

[CR91] D. Chaum, S. Roijakkers: Unconditionally Secure Digital Signatures; Crypto '90, 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 Trans. Inform. Theory 22/6 (1976) 644-654.

[DSS92] The Digital Signature Standard Proposed by NIST; CACM 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, Springer-Verlag, Berlin 1991, 465-473.

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

[HPP92] E. van Heijst, T.P. 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; Proc. IEEE 71/12 (1983) 1341-1345.

[LABW92] B. Lampson, M. Abadi, M. Burrows, E. Wobber: Authentication in Distributed Systems: Theory and Practice; ACM Trans. Comput. Syst. 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 (Ch. 1-3); Laboratory for Computer Science, MIT, Cambridge, MA 02139, USA; distributed at Crypto '91.

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

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

[RSA78] R. L. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; CACM 21/2 (1978) 120-126.

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

[SM93] P. Syverson, C. Meadows: A Logical Language for Specifying Cryptographic Protocol Requirements; Proc. 1993 IEEE Symp. on Security and Privacy, Oakland, California, 165-177.

[Y82] A.C. Yao: Protocols for Secure Computations; 23rd FOCS, IEEE Computer Society, 1982, 160-164.