Abstract appeared in: Proceedings of EUROCRYPT'89, LNCS, Springer-Verlag Heidelberg

The Dining Cryptographers in the Disco: *Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability*

Michael Waidner, Birgit Pfitzmann
Universität Karlsruhe, Postfach 6980
D-7500 Karlsruhe 1, F. R. Germany

Abstract

In Journal of Cryptology 1/1 (1988) 65-75 (= [Chau_88]), David Chaum describes a beautiful technique, the **DC-net**, which should allow participants to send and receive messages anonymously in an arbitrary network. The untraceability of the senders is proved to be unconditional, but that of the recipients implicitly assumes a *reliable* broadcast network. This assumption is unrealistic in some networks, but it can be removed completely by using the fail-stop key generation schemes by Waidner (these proceedings, = [Waid_89]). In both cases, however, each participant can untraceably and permanently disrupt the entire DC-net.

We present a protocol which guarantees **unconditional untraceability**, the original goal of the DC-net, on the **inseparability assumption** (i.e. the attacker must be unable to prevent honest participants from communicating, which is considerably less than reliable broadcast), and **computationally secure serviceability**: Computationally restricted disrupters can be identified and removed from the DC-net.

On the one hand, our solution is based on the lovely idea by David Chaum [Chau_88 § 2.5] of setting traps for disrupters. He suggests a scheme to guarantee unconditional untraceability and computationally secure serviceability, too, but on the reliable broadcast assumption. The same scheme seems to be used by Bos and den Boer (these proceedings, = [BoBo_89]). We show that this scheme needs some changes and refinements before being secure, even on the reliable broadcast assumption.

On the other hand, our solution is based on the idea of **digital signatures whose forgery by an unexpectedly powerful attacker is provable**, which might be of independent interest. We propose such a (one-time) signature scheme based on claw-free permutation pairs; the forgery of signatures is equivalent to finding claws, thus in a special case to the factoring problem. In particular, with such signatures we can, for the first time, realize **fail-stop Byzantine Agreement**, and also **adaptive Byzantine Agreement**, i.e. Byzantine Agreement which can only be disrupted by an attacker who controls at least a third of all participants *and* who can forge signatures.

We also sketch applications of these signatures to a payment system, solving disputes about shared secrets, and signatures which cannot be shown round.

Some younger cryptographers …

are spending the evening in a disco. They are feeling quite relaxed and decide to tell each other who they think are the most fascinating dancers and the best cryptographers among them. However, they haven't lost all inhibitions yet and decide to say their opinions anonymously, remembering that their three bosses once invented a nice protocol for such purposes in a three-star restaurant (Chau_88).

They have some difficulties: The music is loud, and the darkness only broken by flashlights, hence their only way of communicating is to scream into each other's ears. But the admission to the disco was expensive, so they don't want to leave it for a more quiet conversation. They also fear that some of them, afraid that nobody will mention them, might disrupt the conversation. Luckily, their bosses also mentioned ideas of how disrupters could be excluded from the dinner-table.

Now one cryptographer, who thought he was at least a good dancer and therefore did not disrupt the conversation, is excluded. He is so indignant that he leaves the disco and, in the sudden silence outside, notices that the noise may have been the reason for his humiliation. Therefore, he invents a protocol which allows him to simulate the dinner-table broadcast in the disco, which will protect him from such experiences in the future, even if cryptography is wrong and any number of his colleagues are cheaters.

1 Introduction

1.1 Unconditional sender untraceability

In [Chau_88], David Chaum describes a beautiful technique, the **DC-net**, which should allow participants to send and receive messages anonymously in an arbitrary network. (This is the case of a three-star restaurant without disrupters.) In the following, we use the same terms and notations as introduced in [Waid_89].

The basic technique for sender untraceability is **superposed sending**, which realizes an anonymous, additive multi-access channel: For each sending step, called a **round**, each **participant** P_i, i = 1,…,n, chooses a message character from an **alphabet** (a finite abelian group F). Each pair of participants shares one secret **key** (from F, too), which is used with sign "+" by one of them, with sign "-" by the other. Each participant adds his message and all his keys with the appropriate signs. The result is his **output**. All outputs are added, and the **global sum** should be broadcasted to each participant. Since each key has been both added and subtracted exactly once, the global sum is just the sum of all message characters. In particular, if only one of these is $\neq 0$, the global sum is this character.

The untraceability of the senders of messages is proved to be unconditional [Cha3_85, Chau_88, Pfit_89, Waid_89], i.e. the **attacker** may be computationally unlimited, able to eavesdrop communication between any two of the participants, and control an arbitrary subset A of the set of participants P (although at least two honest participants must exist if untraceability is to make sense). However, for recipients the proof implicitly assumes a *reliable* broadcast network. This assumption is unrealistic in some networks, but it can be removed completely by using the fail-stop key generation schemes of [Waid_89]; the resulting net is called **DC⁺-net**.

Superposed sending also works if not all pairs of participants share a key. Those who do can be specified by a graph, called **key graph**.

To use the multi-access channel which superposed sending offers, it is necessary to regulate the participants' access to the channel using an appropriate, i.e. efficient and anonymity preserving protocol. Such a regulation is particularly necessary if one wants to guarantee serviceability, because the easiest way of disturbing the DC- or DC⁺-net is permanent sending. Some protocols of this kind were suggested in [Cha3_85, BoBo_89]; for an in depth discussion of possible protocols cf. [Pfit_89 § 3.1.2].

All our schemes to guarantee serviceability assume that a **reservation technique** is used as the multi-access protocol: A fixed number c of characters is combined into a **message**. Each message is transmitted in c consecutive rounds, called a **slot**. A number of slots for real messages are preceded by a reservation frame, which is used to reserve them for specific participants. The most important reservation techniques, namely reservation map techniques and two techniques using superposed receiving, are also described in [Waid_89].

1.2 Attacks and faults

In both the DC- and the DC⁺-net, however, each participant can untraceably and permanently disrupt the entire net, e.g., by sending in other participants' slots.

Although we will confine ourselves to intentional attacks, it should be noted that the same effects can be caused by physical **faults**. Therefore, on the one hand, in practice a participant identified as an "attacker" should not be removed from the net at once, as in the following protocols, but error recovery should be tried, and in every case he should at least have a second chance. On the other hand, although measures against active attacks also help against faults, most faults should be dealt with more efficiently, especially in layers "under" the DC-net (cf. [Pfi1_85, WaPf_89]).

1.3 Overview

In [Chau_88 § 2.5], the problem of attacks is already discussed, and a scheme for solving it is suggested, which should guarantee untraceability on the reliable broadcast assumption, and serviceability if, in addition, the attacker is computationally restricted. The same scheme seems to be used in [BoBo_89], but with a more efficient reservation technique. (Note that the assumed attacker on serviceability is weaker than the attacker on untraceability, i.e. attackers are conceivable who could disrupt superposed sending without being able to trace messages).

In Chapter 2 we briefly review this scheme. We then show that it needs some changes and refinements before being secure (even on the reliable broadcast assumption). We discuss some of these and propose an improved protocol. It guarantees (as the original protocol ought to) untraceability on the reliable broadcast assumption, and serviceability if, in addition, the attacker is computationally restricted. In an alternative version, both untraceability and serviceability are guaranteed if the reliable broadcast assumption holds and the attacker consists of less than one third of the participants.

In Chapter 3 we remove the reliable broadcast assumption in different ways, since the goal of the DC-net was unconditional untraceability. (That is, whereas Chapter 2 solved the problem of disrupters in the three-star restaurant, the scene of Chapter 3 is the disco.)

In 3.1 we briefly show that if reliable broadcast is implemented by Byzantine Agreement, the appropriate protocol in Chapter 2 guarantees both untraceability and serviceability on the same assumption as the agreement.

In 3.2 we introduce **fail-stop Byzantine Agreement**, i.e. computationally secure Byzantine Agreement with the additional property that, as soon as an unexpectedly powerful attacker forges signatures, all other participants will recognize this. By using it to implement reliable broadcast, the protocol in Chapter 2 guarantees **unconditional untraceability on the inseparability assumption**, i.e. untraceability if the attacker cannot prevent the honest participants from communicating, and serviceability if, in addition, the attacker is computationally restricted.

In Chapter 4 we sketch a solution based on a somewhat different scheme. It guarantees both untraceability and serviceability if the reliable broadcast assumption and $2 \cdot |A| + 1 < n$ hold.

Our protocol for fail-stop Byzantine Agreement is based on **digital signatures whose forgery is provable**. We propose such a (one-time) signature scheme based on claw-free permutation pairs; the forgery of signatures is equivalent to finding claws, thus in a special case to the factoring problem.

Such signatures might be of independent interest. They could be used in cases where it is desirable to shift the risk that the signature scheme is broken from the signers, who bear it in current applications of usual signature schemes, to the recipients of the signatures or a central authority who is responsible for the use of the scheme. In particular, with such signatures we can realize **adaptive Byzantine Agreement**, i.e. Byzantine Agreement which can only be disrupted by an attacker who controls at least a third of all participants *and* who can forge signatures. This will briefly be described in 5.1. Also, fail-stop Byzantine Agreement can be used to implement fail-stop broadcast.

Other possible applications are payment systems, solving disputes about shared secrets, and signatures which cannot be shown round (5.2 - 5.4).

2 A protocol based on the reliable broadcast assumption

It cannot be prevented that a dishonest participant P_i disrupts the DC-net by choosing his outputs not according to the rules. Thus one can only try to discover and punish attackers, finally by removing them from the DC-net. In contrast to faults, attackers cannot be localized in a special localization phase after a disturbance has been detected, because a clever attacker would certainly not disturb during such a phase. Also, localization must not take place for a disturbed slot if someone else might legitimately have tried to send a message in it, because that could lead to this message being traced.

Therefore, in [Chau_88 § 2.5] a protocol for laying "**traps**", i.e. meaningless messages whose only purpose is to be disturbed by an attacker, and prosecuting attackers who are caught in them is briefly suggested. The protocol (like the original DC-net) depends on the assumption of a reliable broadcast network. It is briefly repeated in 2.1.

The idea of traps forms the basis of all following protocols. Nevertheless, the protocol as sketched in [Chau_88 § 2.5] will be shown to be insecure in 2.2: even in refined versions, by a kamikaze attack the sender of a randomly selected message can always be identified. In 2.3, the protocol is improved in a way which preserves untraceability on the reliable broadcast assumption. For some protocol steps, we also present refinements which would be needed for the original protocol, too. Another conceivable variant, based on a suggestion by Chaum, is briefly discussed in 2.4.

Throughout this chapter, we assume the existence of a **reliable broadcast network**, i.e. the attacker cannot manipulate the consistency of broadcast. (This is the case of the three-star restaurant). Thus the achieved untraceability is not really unconditional. Since the original DC-net guarantees untraceability on this assumption, there is no need for fail-stop key generation here (cf. 1.1, [Waid_89]). We also assume that the network allows each participant to determine the origin of each published message unambiguously. The term *"P_i publishes message M"* means that P_i sends M to all other participants using the reliable broadcast network.

How realistic these assumptions are, or can be made, will be discussed in 3.1. As far as serviceability is concerned, the attacker is assumed to be computationally limited.

It is important to note that independent of any assumption on possible attacks on serviceability, for untraceability the attacker is only restricted by the assumption of reliable broadcast. In a certain sense, there are **two different attackers** to be considered simultaneously!

2.1 The original trap protocol

The original protocol [Chau_88] assumes that the reservation map techique with group GF(2) is used, but it can also work with other reservation techniques (cf. 1.1, [BoBo_89]). Each participant P_i must reserve exactly one slot in each reservation phase. Before each reservation phase he decides whether to use this slot to send a real message or to send a trap. If he has decided to send a trap and chosen index x for the reservation, he announces this by publishing an encrypted version of the message *"I will use the slot reserved by index x to send a trap y"* . This message will be called "trap proof" in the following.

Each participant commits to his output for slot s before publishing it. This prevents attacking participants from choosing their outputs depending on other participants' outputs.

If the trap set by P_i is disturbed, P_i publishes the trap proof in clear, together with the encryption key used (and for probabilistic encryption the coin tosses used for encryption). Now, the attacker is **prosecuted**:

Each honest participant P_j publishes his message characters M_j^t and all keys K_{jk}^t used for rounds t of the slot s_x which was reserved by index x. The sums can be compared with the publicly known outputs O_j^t. Hence at least one attacking participant P_a (or an inconsistent key pair) can be detected: Either P_a has correctly published the message character $M_a^t \neq 0$ he really used, although he was not allowed to, or he has modified at least one key K_{aj}^t which he claims to share with an honest participant P_j, which will be detected from $K_{aj}^t \neq K_{ja}^t$.

This procedure of publishing and comparing the secrets of a round t will be called "**investigation** of round t". Disturbances during the reservation phase can be investigated in the same way.

If a notoriously disputed key pair is found, this key pair will be eliminated from the key graph, and participants notorious for attacking will be eliminated from the DC-net. The latter will finally result in a DC-net with a partitioned key graph: one partition of the key graph consists of all attacking, the other of all attacked participants.

2.2 An attack

Unfortunately the trap protocol has a serious *weakness*: Even a computationally limited attacker can forge a trap proof for an arbitrary slot not reserved by him a priori.

For this, the attacker publishes the required encrypted message *"I will use the slot reserved by index x to send a trap y"*, but without reserving that slot, i.e. he chooses an index other than x for sending his 1. Now he hopes that some other participant will use index x. For reservation map techniques, this will happen with probability ≈1/n, if the reservation frame length is $r \approx n^2$, as proposed, and because there are n reservations in each frame.

If it does not happen, the attacker gives up for this time and waits for the next reservation frame. So far, the attack cannot be detected. The attacker can therefore keep trying for an arbitrary period. On average, he will have to wait n/2 reservation frames for success. This is acceptable for him.

If indeed another participant uses index x, the attacker sends the trap message y in the slot s_x corresponding to index x, as announced. Note that the attacker can read the message y* sent by the legitimate user of slot s_x in spite of the superposed y.

The attacker then publishes his trap proof for slot s_x and thus causes the deanonymization of the legitimate user of slot s_x. If this legitimate user has also published a trap proof for slot s_x, it is obvious that one of the two proof publishers was an attacker, but it cannot be distinguished which one. Otherwise, all participants publish their secret keys used for slot s_x, and the legitimate user will be identified during the prosecution. On the one hand he has thus been traced as the sender of message y*, on the other he is unjustly punished as an attacker.

The attack can be made more complicated and more dangerous for the attacker by adding some rules to the prosecution protocol, which try to unmask the attacker by the fact that he has not reserved the slot. In the end, the attacker loses a key each time he attacks successfully (the attack is therefore called "kamikaze"), but he can choose the messages he traces (cf. [WaPf_89]).

If a reservation technique based on superposed receiving ([Pfit_89, BoBo_89], see also [Waid_89]) is used in the same protocol, the same attack is possible in principle. Nevertheless, the situation is much better here: The probability that an honest participant reserves the slot an attacker has chosen decreases exponentially in the length of the reservation frame, in contrast to the reservation map techniques. It also decreases exponentially in the expected amount of computation in the technique of [BoBo_89], cf. [Rab1_80]. Therefore, once it has been noticed that such an attack is possible, the probability that the attacker succeeds can be made arbitrarily small.

Most of this section is described in more detail in [WaPf_89]. Additional attacks, which are only successful if the reservation map protocol with group GF(2) is used, can also be found there. They exploit the fact that this protocol does not enable anybody to decide deterministically whether only one participant has reserved a slot (although this is the case as long as there is no active attack, because in the original protocol it is checked that there are exactly n reservations in each frame). An additional attack on serviceability can be avoided by using another reservation technique and dropping the requirement that reservation frames with collisions are investigated.

2.3 An improved protocol, still based on the reliable broadcast assumption

The following protocol can be seen as an extension of that of [Chau_88 § 2.5]. In particular, the idea of laying traps is maintained, but the trap announcements are linked with the traps in a better way. We also refine several other parts of the protocol. This would be needed for the original protocol, too (e.g., the "palaver phase"). We start with a very short overview, before showing some of the details. Even more can be found in [WaPf_89].

Assume a reservation protocol is used, and that n' of all n participants have reserved a slot successfully. *Each* participant P_i who has reserved a slot s_i sends an encrypted **announcement** in slot s_i, and in slot $n'+s_i$ he sends a **trap** or a **non-trap**, i.e. a real message, according to his announcement (see Fig. 1).

Thus trap announcements and traps are unambiguously linked by their slot numbers, i.e. the attacker cannot forge a trap proof a priori for a slot which will be used by another participant. If announcements are unequivocal, i.e. if there is no message which announces both a trap and a non-trap, the attacker cannot misuse non-trap announcements for initiating the prosecution protocol.

Hence the problem of the original protocol is not posed here.

Call these three phases of the protocol **reservation phase**, **announcement phase**, and **sending phase**, respectively.

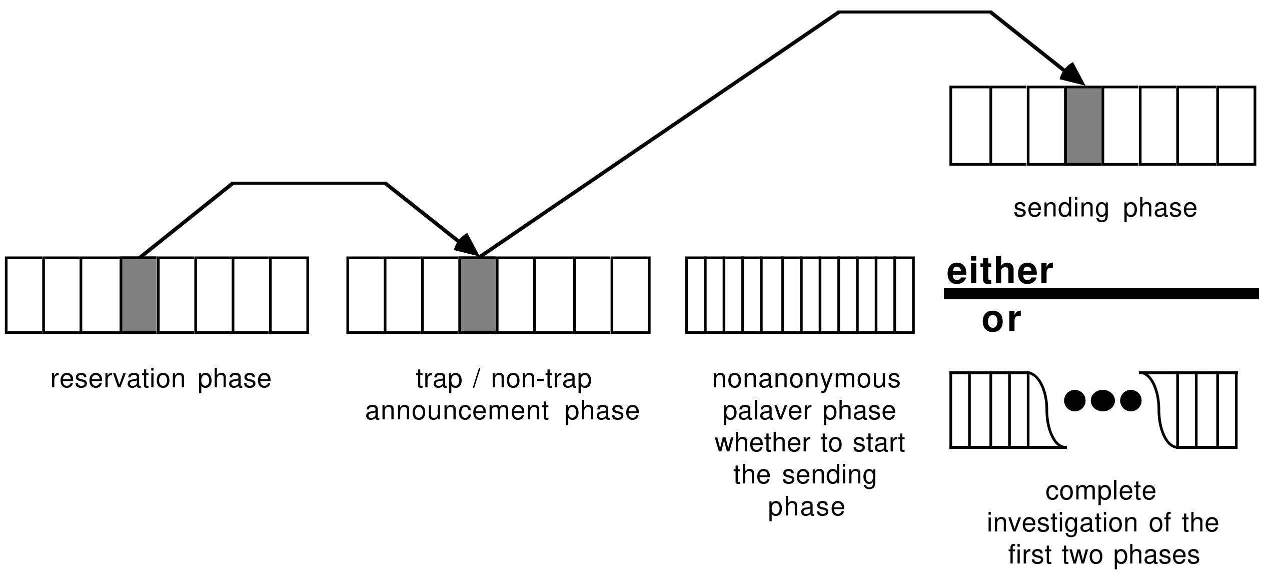

Figure 1 Phases of the improved protocol.

As in the original protocol, the participants' reservation and announcement behaviour is assumed to be independent of their real sending wishes, so that the first two phases may be investigated. Of course, this holds no longer once the sending phase is entered. Hence the reservation and announcement phases and the sending phase must be separated by a non-anonymous **palaver phase**, where each participant who has detected a disturbance during the first two phases can prevent all others from entering the sending phase. Disturbances during the sending phase are only investigated if the corresponding trap proof is given.

Note that if an attacker succeeds in having an honest participant regarded as an attacker and eliminated from the DC-net, this is an attack against untraceability. (One can argue that this participant, now unable to send, is perfectly untraceable. But at least the remaining honest participants are less untraceable than before.) For this kind of attack, the attacker is therefore only restricted by the reliable broadcast assumption. The same holds for the elimination of a key between two honest participants.

1. Outputs and output commitment: All outputs are published using the assumed reliable broadcast network. In all following situations, the number of outputs needed from each participant is known a priori.

We can therefore assume that it has also been decided a priori when each participant has to publish which output (synchronism has to be assumed anyway).

As in the original protocol, for some slots it is necessary to prevent the attacker from choosing his local outputs depending on other participants' local outputs, i.e. each participant has to commit to his local output O_i^t for several rounds before knowing other participants' outputs for any of these rounds. Otherwise, assume that the attacker is the first one who receives all local outputs, i.e. who can compute the correct global sum. In this case, he can disturb the reservation phase by producing collisions, and during the sending phase he can disturb all obviously sensitive messages, e.g., all messages which are addressed to himself.

This output commitment can be carried out using a computationally secure commitment scheme [BrCC_88]. Each participant publishes his commitment in a first phase. In a second phase the commitments are opened, and thus the outputs O_i^t published. Note that an attacker who can break the output commitment scheme in either direction endangers serviceability only. Thus there is no special need to choose a commitment scheme which is unconditionally secure for the prover or the verifier.

Some efficiency improvements are possible. For example, a participant can use a single commitment for all outputs of one slot together. Also, instead of a real commitment scheme, one could try to use the image $f(O_i^t)$ of the output O_i^t under a one-way function f as commitment, as in [Chau_88], but there are one-way functions for which this does not guarantee serviceability [WaPf_89].

A participant who does not output anything when he has to, or whose output does not match his commitment, is considered as an attacker and eliminated from the DC-net. Because of the reliable broadcast assumption, this cannot happen to him if he is honest. Also, the other participants can decide on elimination locally, and all honest participants will obtain the same result.

2. Reservation phase: Since it must be possible to investigate the first two phases without loss of untraceability (before entering the sending phase), no participant should use any sensitive information during the first two phases. Therefore, the reservation behaviour must be independent of the participants' real sending wishes, i.e. some participants will reserve message slots without using them, and others will not be allowed to send all messages they wish.

In [Chau_88], it is suggested that each participant reserves one slot in each reservation phase. To satisfy the independency requirement completely, the ratio of traps and non-traps used by P_i must be fixed, too. This does not, however, guarantee complete untraceability, because if an attacker disturbs a trap, he identifies its sender, and can thus exclude him as a sender of the sensitive messages of the same reservation phase. If this is considered unacceptable, one should use the somewhat less flexible solution that each participant reserves fixed numbers of traps and of non-traps in each reservation phase.

This recommended reservation behavior does not necessarily imply in which situations a reservation phase is investigated, and when a participant is considered as an attacker. However, the easiest way of fixing the *permitted behaviour* is to force each participant to reserve a fixed number u of slots each time.

What happens to *reservation collisions* must also be regulated. All known anonymous reservation techniques (cf. 1.1) are probabilistic, i.e. with each one it can happen that two (honest) participants reserve the same slot. Thus a collision cannot be considered as an attack. To prevent the attacker from disturbing the reservation phase by causing such collisions with honest participants, the reservation phase must be protected by *output commitment*. Except for reservation by superposed receiving with tree-like collision resolution, one should protect a complete reservation frame together.

Output commitment does not prevent attackers from causing collisions among themselves. To avoid the problem of setting a limit for the number of collisions allowed, which always means that an honest participant could be considered as an attacker and eliminated, these should not be considered as an attack either. Since they cannot be prevented either, they must be accepted, i.e. a reservation phase is not investigated just because of collisions. (Thus these attackers only harm themselves).

This enables special attacks if the reservation map protocol with group GF(2) is used (similar to 2.2).

Hence, if one demands that for unconditional untraceability there may not even be a very small chance that an honest participant is regarded as an attacker and eliminated, this reservation protocol should not be used [WaPf_89].

Reservation frames with an *impossible result* (i.e. one which could not have occurred among honest participants) can be investigated, but not all possible checks need to be carried out. To guarantee serviceability, it suffices that each participant checks:

- for the reservation map technique with a large group: the sum of all received reservation messages is equal to u • n
- for superposed receiving: he has received his own reservation messages
- and in the version with tree-like collision resolution additionally: initially, there is a collision between exactly u • n reservation messages (the version of [BoBo_89] enforces this anyway).

During the palaver phase, each honest participant who has seen that one of the tests is not fulfilled will demand an investigation of the first two phases.

All these tests are secure for the tester, i.e. if any of them indicates a fault to the participant who performs it, the investigation will find a protocol deviation. Thus if the investigation does not find any incorrectness, the tester can be assumed to be an attacker.

For each reservation technique, each participant can determine the length of each reservation phase locally (normally it is even constant). Also, each participant knows the number of successful reservations after the reservation phase, i.e. the number of slots of the following phases. Thus if the announcements and messages are of a fixed length, the exact durations of the following announcement and sending phase are determined. (This, or at least some other method by which each participant can locally and information-theoretically unambiguously determine where each announcement and each message ends, is necessary).

3. Announcement phase: Announcements are implemented by using a bit commitment scheme [BrCC_88], i.e. a participant must commit to one of the values "trap" or "non-trap". In contrast to output commitment, the security of the bit commitment scheme used for the announcements has impact not only on serviceability, but also on untraceability: If an attacker can open a non-trap announcement as a trap announcement, he can cause the sender of an arbitrary sensitive message to be identified.

Computationally secure bit commitment schemes can be divided into two classes depending on which part of their definition is unconditionally realized. In our case, one would use a scheme where each announcement can be opened unconditionally in just one way (this is called secure for the *verifier* in [BrCC_88]). Consequently, the indistinguishability of trap and non-trap announcements is only computationally secure.

The use of the opposite class of schemes could be accepted in practice, because if an attacker opened the announcement for a sensitive message as a trap announcement, the real sender would be identified, but he could show his non-trap proof, too, so it would be clear that the scheme is broken.

It is not necessary to protect the sending of announcements by output commitment.

Each honest participant P_i whose announcement was disrupted will demand an investigation of the first two phases during the palaver phase. Since P_i has reserved the corresponding slot, he can prove his right to use the slot during the investigation of the reservation phase. Thus the test is secure for the tester.

4. Palaver phase: Since broadcast inconsistencies are excluded by assumption, the decision to start the sending phase or to investigate the first two phases poses no problem: After the announcement phase, each participant publishes a vote. If at least one participant has detected something wrong and uses his vote to demand an "investigation", the first two phases are investigated completely, and the reservations become invalid. Otherwise the sending phase is entered.

For voting (as for other outputs), there can be a fixed order among the participants. This ensures that each participant has a chance to vote, and that the origin of each vote is clear, so that somebody who disrupts serviceability by wrongly declaring reservation phases as invalid can be punished. The votes should consist of only one bit each. If one wants the participant who detected a disturbance to tell more precisely what it was, this should be postponed to the beginning of the investigation, because otherwise it would unnecessarily reduce the efficiency in the faultless case.

5. Investigation of reservation and announcement phase: If any participants demand an investigation, each honest participant P_i publishes all his message characters M_i^t and secret keys K_{ij}^t for all rounds t of the reservation and announcement phase. The local outputs O_i^t are already known.

Using all these values, each participant can check the behaviour of each other participant and punish the attacker locally. Because of the reliability of the broadcast, and because the check procedure is deterministic, all honest participants will obtain the same results.

First, the rules of superposed sending, and then the rules of the reservation and announcement protocol are checked for each participant, just as in the original protocol. Again, each incorrectness found can be used to eliminate a participant or a key from the DC-net (cf. 2.1, [Chau_88, WaPf_89]).

If no inconsistency is found, the participant P_i who initiated the prosecution protocol is viewed as attacking and eliminated from the net. This is correct, since all the tests which allow a participant to demand an investigation during the palaver phase are secure for the tester.

6. The sending phase and investigation of traps: All slots of the sending phase are protected by output commitment. The commitment must always be made for a complete slot in advance, not for single bits, because an attacker might be able to distinguish traps and sensitive messages after a certain number of bits, e.g., if they are addressed to himself. Hence if an attacker is unable to break the commitment scheme and to distinguish trap and non-trap announcements, and if the ratio of traps to non-traps is fixed to μ, he will disturb traps with probability $\mu/(1+\mu)$.

Slots are only checked if the corresponding trap proof is given, i.e. published an arbitrary (but for practical reasons limited) period of time after the trap was sent. Each participant must have a chance to publish trap proofs, hence some of the bandwidth of the reliable broadcast channel must be reserved for him and this purpose. For example, there can be trap-palaver phases every now and then; the trap proof itself can be given at the beginning of the following investigation.

If a trap proof has been published, first the rules of superposed sending are checked.

Then one has to distinguish between the trapper and the attacker. (It is not clear that the participant who published the trap proof is the trapper: Otherwise, the computationally unlimited attacker on untraceability could decrypt the trap proofs of other participants and publish them first. In this case, the real trapper would be regarded as an attacker and eliminated from the DC-net.)

This can be done most easily if a trapper is forced to send nothing (i.e. zeros) in his own trap. In this case, any participant who sent anything else is an attacker. Although this enables an attacker to distinguish traps and non-traps, it does not reduce serviceability because of the output commitment. Alternatively, the corresponding reservation can be checked, but only if this is possible without investigating other reservations, too. Thus for superposed receiving, each reservation message would have to be repeated alone at the end of the reservation phase, or the announcement would have to be investigated instead.

Our scheme can be modified to ensure both untraceability and serviceability on the assumption $3 \cdot |A| < n$ [WaPf_89]: For this, one has to replace output commitment by simultaneous broadcast [CGMA_85], and the computationally secure commitment scheme by an information-theoretically secure one. Both can be done by using the techniques of [BeGW_88, ChCD1_88]. Since an attacker with $3 \cdot |A| \geq n$ can open non-trap announcements as trap announcements, this protocol cannot guarantee unconditional untraceability.

2.4 Alternative: Opening all trap announcements

A measure different from that in 2.3 against the attack in 2.2, suggested by David Chaum himself [Chau2_89], is to open all trap announcements. This must take place after the sending phase. Now an attacker who announced a trap for a slot, but did not reserve it, is caught if nobody else reserved it either.

However, there is a certain probability that an honest participant does use the slot; in this case, the attack is successful. Thus this measure should not be used with a reservation map technique, only with some kind of superposed receiving, so that this probability can be made exponentially small in log(|F|).

This measure would make two other changes to the original protocol (cf. 2.1) necessary.

Firstly, instead of reserving exactly one slot per reservation phase, each participant must reserve a fixed number of slots for real messages, and slots for traps in addition, preferably also a fixed number. Otherwise, after each sending phase, when the trap announcements are opened, it is clear that the sensitive messages of this phase were not sent by the trappers. This reduces untraceability considerably, because there must be traps in each reservation phase. (Otherwise, an attacker could safely disrupt the corresponding sending phase, because he can see the number of traps in this protocol, in contrast to that in 2.3). In a long correspondence extending over several reservation phases, it would even be possible to identify a sender completely.

Secondly, it is now essential for unconditional untraceability that the trap announcements are made using a commitment scheme where each announcement can unconditionally be opened in just one way (cf. 2.3), in contrast to the one proposed in [Chau_88]. Otherwise, an attacker might be able to open all his trap announcements in such a way that they announce the trap for a slot an honest participant has reserved, because at the time of opening he has seen the reservations.

With these changes and some of the refinements presented in 2.3, this protocol is about as good and efficient as that in 2.3, except that there is always a very small probability that a message is traced.

3 Removing the reliable broadcast assumption

Without the reliable broadcast assumption, the trap and prosecution protocol described in Chapter 2 cannot be applied without giving away *unconditional* untraceability (this is what the young cryptographers forgot):

An unlimited attacker could forge an honest participant P_i's output. He could then use the prosecution protocol to oust P_i from the key graph, or at least to eliminate a key which P_i shares with another participant, who may be honest. Even an honest court (which can be viewed as a very slow implementation of a perfectly reliable broadcast network) cannot distinguish between original and forged messages afterwards. Hence the unconditional untraceability (at least of the other honest participants) would be lost.

In the following, some methods are discussed to combine untraceability and serviceability without the reliable broadcast assumption.

In 3.1, we discuss what can be achieved by known solutions, especially if the reliable broadcast assumption is directly justified physically or by using Byzantine Agreement protocols. Naturally, the untraceability remains "conditional".

3.2 describes a new scheme, using fail-stop broadcast, which guarantees unconditional untraceability on the inseparability assumption (i.e. untraceability if the attacker cannot prevent communication between two honest participants) and computationally secure serviceability.

3.1 What can be achieved with known techniques

First one can try to implement a reliable broadcast network physically. This can be difficult to achieve at all: Since this network is needed for every output, not only in case of dispute, an electrical broadcast medium might be chosen. (If messages are actively forwarded by other stations, reliable broadcast cannot be guaranteed anyway; in particular, this holds for ring- or tree-shaped networks, cf. [Waid_89]). Also, we cannot expect each participant to control a satellite. Thus ground radio and busses are left. Still, disturbances would have to be localized by physical means, and it must be ensured that these, in their turn, cannot be abused to disrupt the broadcast property.

Anyway, a DC-net should not depend on physical peculiarities of the underlying network.

One can therefore try the second usual way of realizing reliable broadcast, Byzantine Agreement protocols [PeSL_80]. The basic network is always assumed to be synchronous. The attacker is restricted either by

$3 \bullet |A| < n$ or by some computational limitations. Which of the two possible restrictions is assumed must usually be decided a priori. (In 5.1, we present a new protocol where this is not necessary.) It is also assumed that the attacker is unable to prevent communication between honest participants. The part that he cannot cut off communication completely must be realized physically. The part that messages which arrive are correct can be realized cryptographically, if one is willing to accept that, with very small probability, the assumed attacker can nevertheless be successful. In this case, one can use authentication codes [GiMS_74, Sim3_88, WeCa_81]. For a survey of lower bounds for reliable broadcast and known solutions, see [Reis_87]. An efficient randomized protocol can be found in [FeMi_88].

In this context, one should also mention the very general techniques for multi-party computations introduced in [GMW_87]. In principle, they can also be used to implement a sender and recipient untraceability scheme.

Instead of a channel with collisions like that of superposed sending, one can realize a channel which transmits all submitted characters $M_1, \ldots, M_n$ in their numerical order. This also hides the relation between the senders and the characters sent. The input phase of such a net (i.e. the subprotocol all participants use to share their secrets M_i among the others) can be made attacker and fault tolerant by standard techniques without loss of untraceability. The transmission (i.e. the computation phase) itself is fault tolerant by construction. Hence most of the reliability problems of superposed sending would not be posed in such a net.

But they cannot guarantee *unconditional* untraceability: Since Byzantine Agreement is a special case of distributed computation, these techniques cannot be more secure than Byzantine Agreement. Thus they cannot be more powerful than the above protocol together with Byzantine Agreement. There are information-theoretically secure techniques, which assume $3 \bullet |A| < n$ [BeGW_88, ChCD1_88, Chau1_89], and computationally secure techniques, which assume $2 \bullet |A| < n$, and that the attacker is computationally restricted (e.g., see [GaHY_88, GMW_87]). A forthcoming work only has to assume the disjunction of these two assumptions [Chau1_89]. In all cases, the attacker is assumed to be unable to prevent honest participants from communicating.

3.2 Serviceability in spite of unconditional untraceability on the inseparability assumption

Before describing the new protocol which guarantees unconditional untraceability on the inseparability assumption (in particular with no condition on the number of attackers) and computationally secure serviceability, we first suggest an informal definition for guaranteeing serviceability while preserving untraceability. An implementation of the DC-net, together with a protocol to remove attackers, is said to guarantee untraceability in spite of an attacker who is limited by a predicate $\mathbf{A_{untr}}$, and serviceability in spite of an attacker who is limited by a stronger predicate $\mathbf{A_{serv}}$, if the following two conditions hold:

S1 *Serviceability:* If A_{serv} is satisfied (thus A_{untr}, too), the attacker cannot prevent any honest participant from sending a message successfully within a given period of time, without at least losing a key from the key graph.

S2 *Preservation of untraceability:* If the attacker satisfies A_{untr}, and if due to the protocol two honest participants lose a common key from the key graph (or an honest participant is eliminated altogether), each honest participant definitely stops superposed sending.

For example, the protocols in 2.3 guarantee untraceability for A_{untr} = "reliable broadcast assumption", and one of the protocols guarantees serviceability for $A_{serv} = A_{untr} \wedge$ "the attacker is computationally restricted", and the other one for $A_{serv} = A_{untr} \wedge 3 \bullet |A| < n$.

Condition S2 describes a *fail-stop* property. One may therefore consider it to be unnecessarily weak, since, e.g., the protocol in 2.3 satisfies the stronger and more natural condition

S2* If the attacker satisfies A_{untr}, two honest participants will never lose a common key from the key graph (nor will an honest participant be eliminated altogether).

However, this makes no real difference: If A_{untr} holds, but not A_{serv}, serviceability is not guaranteed anyway, thus another reason for the honest participants to stop participating is not harmful. If both A_{serv} and A_{untr} hold, $S2^*$ is implied by S1 and S2 anyway.

In 3.2.3, we will fulfil S1 and S2 by using a new kind of Byzantine Agreement together with the protocol in 2.3. For this, in 3.2.2 we present an idea which transforms each Byzantine Agreement protocol which works with any kind of signatures (all computationally secure Byzantine Agreement protocols we know do this) for tolerating up to n-2 attackers into a protocol which

- guarantees agreement provided the attacker is not more powerful than assumed, and
- guarantees unconditionally that all honest participants stop participating, once the signature scheme is broken.

This is achieved by a new signature scheme (cf. 3.2.1) which, with very high probability, allows participants whose signatures are broken to prove this to all other participants in an unconditional way. Hence, soon after the first signature of an honest participant's is broken, all honest participants can stop their participation with the same very high probability (**fail-stop Byzantine Agreement**).

For guaranteeing the fail-stop property, it is only necessary to assume the attacker to be unable to prevent communication between honest participants. This seems to be the weakest possible assumption, and is called **inseparability assumption**.

Parts of the following can be found in more detail in [WaPf_89], the rest is to be fully formalized in [Pfit1_89].

3.2.1 A signature scheme where forgery can be proved

Before describing our signature scheme and its properties, some remarks about the principal possibilities of such a scheme can be made:

- The probability of unprovable forgery can never be zero: An honest participant (Alice) must be able to produce at least one signature for each message, thus if a forging attacker (Felix) finds just this signature, Alice must be unable to prove that it is not hers.

- There must be more than one possible signature for each message, i.e. more than one value must be acceptable to the other participants (Bob_i). This must hold unconditionally.

  This is for the same reason as the previous point: Only a signature which Alice could not have produced herself may enable her to prove that it is not hers; and the computationally unlimited Felix must be unable to distinguish them.

- It is not possible to prevent a dishonest, computationally unlimited participant from denying her signatures, i.e. "proving" that they were broken, even though this is not true. But in all our applications this is entirely right.

Thus one is looking for a signature scheme where most forged signatures enable the supposed signer to compute something which, on the assumption of the signature scheme, she could not have computed before.

The idea of the following scheme is to use a function g where every image has several preimages, but which is collision-free in the computational sense. Signatures consist of preimages of previously known values. Forged signatures usually lead to a collision with the correct one. A collision serves as **proof of forgery**.

To ensure that a signer cannot produce "proofs of forgery" if no forgery has occurred, she must not be allowed to choose the function g herself. This is especially clear if g has a trap-door, but in every case, one-way properties are only guaranteed against people who only receive the description of the function used. More precisely, the easiest way of convincing Bob_i that a collision is really the result of someone breaking the scheme is that he has chosen g himself.

Therefore we obtain a system very similar to one-time signatures, which are an easy, but rather inefficient method for signing a limited number of bits using a one-way function, attributed to Lamport in [DiHe_76, Merk_88]. (The one-way function in our case is g.) Also, some of the efficiency improvements to one-time signatures by Merkle can be adapted so that forgery can be proved.

We will first sketch a possible choice of functions g, and then describe the overall scheme in more detail. An easier, but less efficient scheme based on modular squaring as the function g can be found in [WaPf_89]. (The efficiency improvements to that scheme, presented there in § 4.4.3.1.3, are incorrect.)

3.2.1.1 Hiding functions

One (not the only) way of obtaining g's for which the probability that a forgery leads to a proof of forgery is at least $1\text{-}2^{-\sigma}$ for some σ chosen a priori, is to use a family of functions with the following characteristics: On input 1^k, a generator G outputs two polynomial-time algorithms d, g such that

- d defines a uniform distribution over D := d( )
- for each $x \in \{0,1\}^\sigma$, g(x, •) is a permutation of D
- for any probabilistic polynomial-time algorithm, the probability that it finds a collision (in $\{0,1\}^\sigma \times$ D), if g is chosen by $G(1^k)$, decreases faster than 1/p(k) for any polynomial p.

We call the g's **hiding functions** (because they can be used to hide the first argument; e.g., as a commitment scheme unconditionally secure for the prover). Similar functions have been used for other purposes before, and Bert den Boer is said to have proposed some whose security is reducible to factoring or to discrete log [ChFN_88]. So we will just present a construction based on any family of claw-free permutation pairs (not necessarily trap-door), not caring whether efficiency can be improved. It is very similar to the construction in the GMR signature scheme [GoMR_88], where claw-free pairs are defined, and to the hash functions in [Damg_88]: Just use the generator for claw-free pairs of permutations, and from each pair (f_0, f_1), derive the function g defined by

$$g(x,y) := f_x (y) := f_{x_1} (\dots (f_{x_\sigma} (y))\dots).$$

The proof that these g's are collision-free is similar to those in [GoMR_88, Damg_88]; note that no additional prefix-free encoding is necessary, since the length of x is fixed. In particular, by using claw-free pairs of [GoMR_88, Damg_88], one can obtain g's whose collision-freeness is equivalent to factoring or the discrete log.

3.2.1.2 Efficiency improvements

If the functions g were to be used as in the one-time signature scheme, to sign Λ bits, Alice would have to do the following: In advance, she chooses $2 \bullet \Lambda$ pairs $(x_{\lambda,b},y_{\lambda,b})$ randomly, where $\lambda=1,\dots,\Lambda$, b=0,1. She publishes the images $g(x_{\lambda,b},y_{\lambda,b})$. Later, as a signature that the value of the λ-th bit is b, she shows the preimage $(x_{\lambda,b},y_{\lambda,b})$.

1. The main disadvantage of this scheme is that a large number of values must be published in advance. This can be avoided by tree authentication like that by Merkle for publishing lists of public keys [Merk_80]: Let h be a hash function. The values to be published are used as leaves of a binary tree. The value of each inner node is h(left_child, right_child). Only the value of the root is published. Later, to authenticate one of the leaf values, all its forefathers and the other child of each one must be shown.

   Again, a collision will serve as proof of forgery.
2. For 1., the depth of the tree, and thus the number of bits which can be signed, must be fixed a priori. This can be avoided by using some of these bits as roots of new trees. Depending on the application, this can be done in two ways.

   One way is to form a global tree structure like that in [Merk_88]. To distinguish the trees and signatures used in 1. from this global structure, we will call them mini-trees and mini-signatures for the moment. The nodes of the global tree are the roots of the mini-trees. R_2 is a child of R_1 in the global tree iff R_2 is authenticated within the mini-tree of R_1. Each global signature consists of one mini-signature of the message with respect to one mini-tree root R, and the mini-signatures of R and all its forefathers in the global tree. However, here one would choose the mini-trees much larger than in [Merk_88], where each one only serves to sign one message and two new roots.

The second is to exchange the roots completely, broadcasting the new roots using the protocol in 3.2.2. In contrast to the first way, old mini-trees need not be stored, and the length of signatures remains constant. This is particularly suitable for applications where many signatures are needed, but need not be stored for a long time, e.g., reliable broadcast.

3. Like in [Merk1_82] (according to [Merk_88]), only bits with the value 1 need to be signed if the messages are first encoded by a code which detects all changes of 1 into 0.
4. If the messages to be signed are long, they can be hashed first, and only the result is signed bit by bit. Again, a collision must be accepted as proof of forgery.

Note that the hash functions used in 1. and 4. must, of course, be collision-free in the computational sense, but that no additional information-theoretical requirement, like for the functions g, is necessary, since it does not matter if the preimages are guessed. Therefore any hash functions can be used, e.g., those in [Damg_88]. These are particularly suitable here, because they are based on the same intractability assumptions as the hiding functions.

3.2.1.3 The complete scheme

The resulting signature scheme can best be presented in protocol form. One can first construct a two-party protocol for a signer Alice and a recipient Bob [WaPf_89], and any outsider Vera can later solve disputes between them. Here we will describe the full n-party protocol, where each party wants to sign messages and to verify all other signatures, at once.

Assume that a generator G for hiding functions and a generator H for hash functions have been chosen. Let k be the **security parameter for collision-freeness**, i.e. the value for which it is assumed that finding collisions of g or h is infeasible for the attacker. A second security parameter σ, the **security parameter for the probability of unprovable forgery**, is chosen.

For simplicity, we assume that each participant wants to sign a fixed number N of messages, and that either the length of each message is fixed a priori (this might be the case in the reliable broadcast application), or they are first hashed to a fixed length. Let C be the encoding function used for efficiency improvement 3, and L the length of the messages when they are encoded using C.

The protocol needs one step more than explained so far: Since the recipients of signatures are allowed to choose the functions, it must be ensured that these functions have the properties which are necessary for the signers. In particular, $g(x, \bullet)$ must really be a permutation of D, and d must define a uniform distribution on D. This can be shown by a zero-knowledge proof that (d, g) is indeed a possible output of $G(1^k)$. (It must be a proof in the unconditional sense, except perhaps with an exponentially small chance of cheating, whereas the zero-knowledge need not be perfect.) If P_i and P_j disagree on the outcome of this proof, each other honest participant P_m must be able to decide which one is lying. (This does not mean that P_m is convinced by the proof if P_i and P_j agree on it.) For example, the proofs in [GoM1_86] have these properties. If the hiding functions are based on the special claw-free pairs in [GoMR_88 § 6.3], the proof in [GrPe_88] for Blum integers should suffice.

During the key distribution phase, we must of course assume some non-cryptographic form of reliable broadcast.

The resulting protocol, not in a fully formalized form, looks like this:

Key distribution phase:

[1] Each participant P_i uses $G(1^k)$ to choose d_i and g_i, and $H(1^k)$ to choose h_i, and publishes them. (If, e.g., G and H are based on the same family of claw-free pairs, one can also choose just one claw-free pair and construct both g_i and h_i from it.)

[2] For each pair P_i, P_j of participants, P_i publicly gives P_j a zero-knowledge proof that g_i,

together with d_i, is a correct hiding function. If P_i does not succeed, each other participant P_m considers P_i as an attacker.

[3] For each P_i, participant P_j does the following:

For $\mu = 1,\ldots,N$, $l = 1,\ldots,L$, P_j chooses $x^{(i,j)}_{\mu,l} \in \{0,1\}^\sigma$ randomly, and $y^{(i,j)}_{\mu,l}$ using d_i, and stores them.

She forms the images $g_i\,(x^{(i,j)}_{\mu,l}, y^{(i,j)}_{\mu,l})$, and uses them as leaves of a binary tree $T^{(i,j)}$. The value of each inner node is computed by applying h_i to its two children.

P_j makes the value $R^{(i,j)}$ of the root public.

Main phase:

[4] Assume that as her μ-th signature, P_j wants to sign a message M. The signature consists of n parts $S_1,\ldots,S_n$, where S_i is meant to convince P_i.

To form S_i, let $M_i^* = C(M)$ or $M_i^* = C(h_i(M))$, depending on whether the messages are of fixed length or must first be hashed. Let $b^{(i)}_1,\ldots, b^{(i)}_L$ be the bits of M_i^*.

P_j looks up the values

$$u^{(i)}_l := (x^{(i,j)}_{\mu,l}, y^{(i,j)}_{\mu,l}) \text{ for all } l \in \{1,\ldots,L\} \text{ with } b^{(i)}_l = 1$$

S_i consists of the $u^{(i)}_l$'s, together with the values to check their authenticity with respect to $R^{(i,j)}$, i.e. the other children of all the forefathers of the $g_i\,(u^{(i)}_l)$'s in $T^{(i,j)}$, in some predefined order.

P_j stores (M,μ).

[5] Each P_m who receives a message M, the numbers j of the signer and μ of the signature, and such a signature $(S_1,\ldots, S_n)$, must check each part S_i. For that, P_m first computes $b^{(i)}_1,\ldots, b^{(i)}_L$ just as in [4].

P_m now uses g_i and h_i to check whether the $g_i\,(u^{(i)}_l)$'s and the remaining components of S_i form the correct branches of a tree with the root $T^{(i,j)}$.

Forgery check phase:

[6] Each honest participant who receives a signature of P_j's, sends it back to P_j.

[7] Assume that P_j receives a forged signature $(S_1,\ldots, S_n)$ for a message M and a number μ. For each P_i, P_j tries to find either a g_i- or an h_i-collision. This is carried out by comparing the forged with the corresponding correct signature: First, if messages are hashed and P_j has already used μ to sign a message M', P_j checks if $h_i(M) = h_i(M')$. If not, P_j checks if $u^{(i)}_l = (x^{(i,j)}_{\mu,l}, y^{(i,j)}_{\mu,l})$ for all components $u^{(i)}_l$ of S_i, and if all other components have the value of the corresponding node in $T^{(i,j)}$. If not, there must be a collision with the correct signature on the path from the incorrect node to the correct root.

P_j publishes these collisions.

[8] Each honest participant P_m who receives a g_i- or h_i-collision accepts this as **proof of forgery for P_i**. If there is a proof of forgery for P_i for *all* P_i, P_m accepts this as **proof of forgery** (more precisely: as proof that someone can break the signature scheme, not that someone has forged anything).

It is not yet obvious why we emphasized the fact that participant P_m waits to see a proof of forgery for each participant, and not just for himself, before he accepts it as proof of forgery. This will be needed when the scheme is used within Byzantine Agreement, because this fact serves to ensure agreement upon whether the scheme was broken or not. This also explains why P_j's signatures must contain a part S_j, at least if the function g_j is not trap-door: A forged signature must, in particular, guarantee that P_j finds a collision of her own functions.

Formalizing the reasons which were used as motivation for the scheme, one obtains the following lemma:

Lemma 1 Assume the basic protocol for n parties described above, and that no undiscovered cheating has occurred during phase [2].

i. The problem of forging a signature which an honest participant will accept in step [5] is equivalent to breaking one of the families of collision-free functions.

ii. Each forged signature enables the supposed signer P_j to form a proof of forgery, with probability not less than

$$\left(1 - \frac{1}{2^{\sigma}}\right)^{n} .$$

iii. Conversely, forging a proof of forgery is equivalent to breaking one of the families of collision-free functions.

A sketch of a proof of the previous simpler scheme can be found in [WaPf_89], a complete proof is to appear in [Pfit1_89].

The length of the signatures can be reduced, if the participants agree upon one g and one h, instead of all the g_i, h_i. This may be achieved by using a computationally secure protocol for multi-party computations: The random inputs q, r for G, H are chosen by all participants together, e.g. as the sum of values q_i, r_i which are chosen by P_i randomly. To execute the whole algorithm (i.e. G, H, and the computation of q, r from q_i, r_i), the multi-party protocol of [ChDG_88], with a bit commitment scheme unconditionally secure for the verifier, is used. Since the original generators G and H are errorless, i.e. for all inputs they compute correct functions g and h, the output is unconditionally correct with an exponentially small error probability. If the attacker is unable to break the commitment scheme, he obtains no trap-door information about g, h.

Each participant who disrupts the computation by submitting wrong messages (which will be recognized with an at most exponentially small error probability) is assumed to be an attacker, and the computation is repeated without him. (Usually, e.g., in [GMW_87, GaHY_88], it is suggested that a disturbing participant is somehow simulated by the other participants, i.e. disturbances are *tolerated*. For this, each participant's secret must be shared among the others, thus one has to assume $2 \cdot |A| < n$ additionally. In our case, there is nothing to gain from forcing a dishonest participant to take part.)

Anyway, these protocols are not very practical. The situation seems better if the (otherwise less efficient) collision-free pairs based on the discrete log of [Damg_88] are used, because in that case, any protocol for choosing mutually trusted random bits should suffice.

3.2.2 A protocol for fail-stop Byzantine Agreement

In [DoSt_83], Dolev and Strong describe (and prove) a computationally secure protocol for Byzantine Agreement, which can tolerate any number of attackers and where each honest participant needs to send two different messages at the most. If there is only one or no honest participant, there is no problem, hence we assume $|A| < n-1$ in the following.

We derive a fail-stop Byzantine Agreement protocol from that specific protocol, although a similar construction is possible from any Byzantine Agreement protocol which works with signatures. Each message of the protocol consists of the value the sender has broadcasted, and a maximum of n-1 signatures. Let $s_i(v)$ denote the value v signed by P_i, and assume that from $s_i(v)$ the signed message v can be extracted.

The original protocol uses iterated signatures (like $s_{i_w}(s_{i_{w-1}}(\ldots s_{i_2}(s_{i_1}(v))\ldots))$). In our signature scheme, signatures are much longer than messages. The computational expenditure of the signing process and, if the messages are so short that they need not be hashed, also the length of the signatures grow with the length of the message. It is therefore preferable to use *sets* of signatures instead (like $\{s_{i_w}(v), s_{i_{w-1}}(v), \ldots, s_{i_2}(v), s_{i_1}(v)\}$). (For the signature scheme in [WaPf_89], iterated signatures would even grow exponentially with the number of signers.)

Luckily, the Dolev/Strong protocol can also be implemented with sets of signatures. (Because of this change, and since the protocol description given in [DoSt_83] is rather short anyway, we have presented the proof by Dolev and Strong for our scheme in [WaPf_89]).

From the Dolev/Strong protocol, we obtain an (n+1)-phase protocol for Byzantine Agreement where breaking can be proved, in the following way:

Byzantine Agreement protocol where breaking can be proved, for up to n-2 attackers

For $\phi = 1,\ldots, n-1$:

[ϕ] Perform the ϕ-th phase of the Dolev/Strong protocol, but using sets of signatures instead of iterated signatures, and using the signature scheme in 3.2.1.

[n] Each participant P_i sends all signatures received during the previous phases back to their supposed signers, i.e. he sends $s_j(v')$ back to P_j.

[n+1] If P_j has received a forged signature $s_j(v')$, she tries to form a proof of forgery. If she is successful, she sends it to all other participants.

[End] If P_i has not found or received a proof of forgery, he decides as in the original Dolev/Strong protocol. Otherwise he decides "signatures broken".

Lemma 2 **Byzantine Agreement where breaking can be proved.** Assume the above protocol is executed, $|A| < n-1$, and that the attacker is unable to prevent communication between honest participants.

i. If the attacker is unable to forge signatures (i.e. to break the collision-free functions), the protocol finds the correct agreement after phase [n+1].

ii. Assume that no undetected cheating occurred during step [2] of the signature scheme.

If an honest participant decides to accept the value v after phase [n+1] (according to the Dolev/Strong decision rule), then with probability at least

$$\left(1 - \frac{1}{2^{\sigma}}\right)^{n},$$

v is the correct value (i.e. the sender's value if the sender was honest), and each other honest participant has either accepted value v, too, or has decided "signatures broken".

iii. If an honest participant decides "signatures broken", he can prove this to all other participants after the protocol.

Each protocol which satisfies (i), (ii), and (iii), is called **Byzantine Agreement where breaking can be proved**.

Proof. i. Since the attacker cannot forge signatures, the protocol works like the original Dolev/Strong protocol. Agreement is therefore guaranteed.

ii. Assume that the honest P_i has accepted v after phase [n+1].

In contradiction to Lemma 2, assume that v is *not* the correct value. Since the Dolev/Strong protocol always finds the correct value, provided signatures of honest participants are authentic, there must be at least one honest participant who has received a forged signature of an honest participant P_j's. Thus in phase [n], P_j receives the forgery, and in phase [n+1], she sends the proof of forgery to all other participants with probability $(1-2^{-\sigma})^n$. Hence in phase [n+1], P_i receives the proof of forgery and decides "signatures broken". This contradicts our assumption. Thus P_i has accepted the correct value v.

Since the same is true for all other honest participants, each one who has not decided "signatures broken" has accepted the same value v.

iii. If an honest participant decides "signatures broken", he knows a proof of forgery and can show it to all other participants later. □

This protocol can be described as realizing Byzantine Agreement if the signatures are not broken, and Crusader Agreement [Dole_81] otherwise. Additionally, the participants who know that something is wrong can prove it later.

Each participant has to send two messages at the most, and, in particular, to sign a maximum of two values.

Now assume that the above protocol is repeated ad infinitum, i.e. that after the last phase a new phase [1] starts. Call each protocol execution a **broadcast**. For each broadcast, it is predetermined which participant is the sender. We add the following supplement to the protocol:

Supplement to the Byzantine Agreement protocol

For each phase [ϕ], add the following rule to the protocol:

[ϕ] If in phase [ϕ-1], P_i receives a proof of forgery for the first time, he immediately decides "signatures broken" for *all* following (and the current) broadcasts, and sends the proof of forgery to all other participants. No participant who has decided "signatures broken" will ever send any further messages.

This supplement guarantees that, if an honest participant decides "signatures broken" in phase [ϕ], each other honest participant does so in phase [ϕ+1]. Hence we have realized fail-stop Byzantine Agreement.

Lemma 3 Fail-stop Byzantine Agreement. Assume that the above protocol with the supplement is iterated ad infinitum, $|A| < n-1$, and that the attacker is unable to prevent communication between honest participants.

i. If the attacker is unable to forge signatures (i.e. to break the collision-free functions), the protocol realizes reliable broadcast.

ii. Assume that no undetected cheating occurred during step [2] of the signature scheme.

If an attacker, who can forge signatures, disturbs the τ-th broadcast during the first n phases, then with probability at least

$$\left(1 - \frac{1}{2^{\sigma}}\right)^{n},$$

all honest participants decide "signatures broken" at the end of the τ-th broadcast, and stop sending.

iii. If an attacker, who can forge signatures, disturbs the τ-th broadcast during phase [n+1], each honest participant either decides "signatures broken" at the end of the τ-th broadcast, or accepts the correct value at the end of the τ-th broadcast and stops sending after phase [1] of the (τ+1)st broadcast.

Proof. i. Follows from Lemma 2(i).

ii. The first two cases are that the attacker sends the forged signature of an honest participant P_j's either to another honest participant in a phase $\phi < n$, or directly to P_j in phase [n]. Thus with the above-mentioned probability, each participant receives the proof of forgery from P_j in phase [n+1] of the τ-th broadcast, decides "signatures broken", and stops sending.

The third case is that he sends a proof of forgery to an honest participant directly in a phase ϕ for the first time. Then in phase $\phi+1 \leq n+1$, all other honest participants receive it, decide "signatures broken", and stop sending.

iii. In this case, the attacker sends a proof of forgery to an honest participant directly in phase [n+1] for the first time. Thus the decisions according to the original Dolev/Strong protocol are still correct, and each honest participant receives the proof of forgery at the latest in phase [1] of the (τ+1)st broadcast. □

3.2.3 Using fail-stop Byzantine Agreement for untraceability and serviceability

Together with the protocol in 2.3 (i.e. used in all places where the reliable broadcast network was needed), fail-stop broadcast guarantees computationally secure serviceability while preserving unconditional untraceability on the inseparability assumption (i.e. the attacker on untraceability is only assumed to be unable to prevent communication between honest participants) in the sense of S2:

The only way two honest participants can decide differently is that one has accepted a value v while the other has decided "signatures broken" (Lemma 2 ii). But in this case, all honest participants will stop after the first phase of the next broadcast (Lemma 3 ii, iii), and those who perform the first phase of the next broadcast have accepted the correct value. Thus the attacker cannot learn more than an additional output O_i^t from a participant who has not received any incorrect value. This cannot give him additional information.

Since this scheme is much less efficient than the schemes of [Waid_89], in practice one could try to use one of those, and only if that were disrupted continually, would one switch to the scheme described here (by hand, because first the key distribution phase would have to be performed).

4 A protocol without commitment

In the protocols described so far, the assumption that the attacker on serviceability must be restricted by $3 \cdot |A| < n$ or computationally limited (in addition to the reliable broadcast assumption) were needed for commitment.

In this chapter we sketch a protocol without commitment, which only assumes reliable broadcast and $2 \cdot |A| + 1 < n$. It has a similar phase structure as that in 2.3, although the non-traps are distinguished from traps in another way.

The idea is that a commitment is not really necessary for trap/non-trap announcements (because if the owner changes his mind about whether it is a trap or not, this does not harm anybody else, because the others are not allowed to disturb it anyway). The problem that only the owner may be allowed to announce that it was a trap is solved by a password scheme: In the announcement phase, each other participant sends the anonymous owner of a slot a password. The sending of passwords is protected by keyless cryptography [AlSc_83, Pfit_89] and keyless authentication [Bura_88, WaPf_89], which can be implemented very efficiently using superposed sending.

Output commitment is no longer possible on this assumption either. Instead, the participants must send in a different order in each protocol execution.

A full description can be found in [WaPf_89].

5 Other applications of the signature scheme

5.1 Adaptive Byzantine Agreement

The signature scheme in 3.2.1 can also be used for a Byzantine Agreement protocol where one need not decide between the two restrictions on the attacker (computationally restricted or $3 \cdot |A| < n$) a priori. Instead, it will always work if at least one of them is fulfilled. (A similar protocol will appear as a special case of the protocol in [Chau1_89], but it seems to need the assumption $2 \cdot |A| < n$ in addition to the computational restriction for fault tolerance.)

The first idea is, of course, to start with a Byzantine Agreement protocol BA1 which works with any kind of signatures, implemented with the signature scheme where breaking can be proved, and to have another Byzantine Agreement protocol BA2 which works for $3 \cdot |A| < n$ as an alternative.

Now, in contrast to the situation in 3.2, one needs a definite moment when everybody knows whether the signatures are assumed broken and BA2 must be used, or whether the results of BA1 are used. But in a suitable implementation, this doubtful situation can only arise if the signatures are broken. Thus luckily, here we can assume that the attacker is restricted by $3 \cdot |A| < n$ (also in contrast to 3.2, where we had to consider the unlimited attacker on untraceability).

This provides motivation for a simple protocol where, after the execution of BA1 and BA2 for the original value, each participant uses BA2 to broadcast whether he knows a proof of forgery. If he does, he must broadcast not just the value "yes", but the proof of forgery itself. This prevents the participants from disturbing the decision if $3 \cdot |A| < n$ does not hold (but, consequently, the computational restriction does).

The exact protocol and its proof can be found in [WaPf_89], together with some simple efficiency improvements, e.g., to $\lceil 4/3 \cdot n \rceil$ phases.

5.2 Payment systems

Signatures where forgery can be proved can be used to shift the risk that the system is broken, which is usually borne by the signer, to the recipient of the signature. (This, however, is not necessary; one can also leave the risk with the signer, but limit it by the fact that the system is no longer used once the first proof of forgery appears.)

This could, e.g., be useful with the clients' signatures in a payment system. Since each client only signs messages for the bank, the basic two-party protocol (cf. 3.2.1.3) suffices. For example, in [ChFN_88 § 2] it was regretted that the scheme is not unconditionally secure for the clients. This can easily be achieved: the scheme already contains preimages of a hiding function as the clients' signatures. Only, the images must be authenticated using the tree authentication in 3.2.1.2, instead of a normal signature scheme. (Since in this case the clients sign one-bit messages only, and only one pair g and h, chosen by the bank, needs to be used, the size of the signatures is comparable to that of normal signature schemes.)

5.3 Solving disputes about secret keys

Another useful application, suggested by David Chaum [Chau3_89], is solving disputes about shared secret keys, e.g., during an investigation into some rounds of the DC-net (cf. [Chau_88] or 2.1). The original protocol in [Chau_88 § 2.6] is just that the two participants exchange signatures on the shared key bits or on "split bits", i.e. two values whose exclusive or is the key bit, and to use tree authentication (cf. [Merk_80] or 3.2.1.2) as an efficiency improvement. This can easily be adapted to yield unconditional security, without losing too much efficiency, because only two parties are involved, and the values to be signed are already known during what corresponds to the key distribution phase. The basic addition is, of course, that collisions of the hash function used in the authentication tree serve as proof of forgery. However, some change is also necessary in the authentication of the root R:

The way with the smallest change is not to use split bits, and to add the rule that if the two participants show (normal) signatures of each other on different values for R, the conflict cannot be resolved (because either both are attackers or one has forged the signature of the other). Thus in the case of the DC-net, one would return to removing the key. (In the case of split bits, there would be no way at all of recognizing forged signatures.)

A second way is to substitute the top level signature by one whose forgery can be proved. Now, if no split bits are used and the participants show signatures on different values for R, both can be considered as attackers (since if only one were an attacker, one of the signatures would have to be forged, thus the other one could prove it).

A third way is to make R public. Of course, this must not undermine the unconditional secrecy of the key. For this, instead of R itself, a hiding function of it could be published, i.e. g(R, y) for a random y.

If one wishes that in the course of solving a dispute about some bits, no information about other bits becomes public (this is a scenario where not all attackers collude), the intermediate hash values in the authentication tree must not contain such information. Therefore one should apply a hiding function to the leaves of the authentication tree, i.e. to the blocks of bits which are always investigated together.

5.4 Signatures which cannot be shown round

Finally, another property, which has just been noticed as desirable for signature schemes in some applications [ChAn_89], is somehow inherent in our scheme. This is that the recipient of a signature, Bob, cannot show it around without the help of the signer, Alice (who would sometimes be forced to provide this help, e.g., if asked to do so by a court).

For this, the two-party protocol must be used, where each signature uses just one g and one h, chosen by Bob. The hiding function g must be based on claw-free pairs with trap-doors, as in 3.2.1.1. Bob must be told the leaves of the authentication tree. Now, using the trap-door, he can form valid looking signatures himself. They only become authentic if Alice declares that she cannot find a proof of forgery against them.

The difference to the scheme in [ChAn_89] is like that to usual signature schemes: that scheme is not unconditionally secure for the signer, but the signatures are usually much shorter there.

6 Summary

The goal of this paper was to investigate how, on the basis of superposed sending (1.1), a sender and recipient untraceability scheme can be realized which additionally guarantees serviceability on certain assumptions.

The protocol for solving this problem suggested in [Chau_88] can be misused for an easy active attack on untraceability (2.2), but its basic idea can be used for the secure solutions presented.

The restrictions A_{serv} and A_{untr} on the attackers against serviceability and untraceability, resp., which are necessary for the different protocols, are summarized in Figure 2. For a comparison, we include the schemes of [Waid_89]. P is the set of participants, A the set of attackers.

<table>
<tr><th>Protocol</th><th>Section</th><th>A_{serv}</th><th>A_{untr}</th><th>Untraceability probabilistic</th></tr>
<tr><td>Superposed sending</td><td>1.1, [Chau_88]</td><td>$A = \emptyset$</td><td>reliable broadcast</td><td>no</td></tr>
<tr><td>Fail-stop broadcast</td><td>[Waid_89, 3.2.2.1]</td><td>$A = \emptyset$</td><td>unconditional</td><td>no</td></tr>
<tr><td></td><td>[Waid_89, 3.2.2.2-3]</td><td>$A = \emptyset$</td><td>unconditional</td><td>yes</td></tr>
<tr><td></td><td></td><td colspan="3">reliable broadcast assumption</td></tr>
<tr><td></td><td>2.3</td><td>$\wedge$ comp. restr.</td><td>$\wedge$ true</td><td>no</td></tr>
<tr><td>Reliable broadcast</td><td>2.3</td><td>$\wedge\ 3 \cdot |A| < |P|$</td><td>$\wedge\ 3 \cdot |A| < |P|$</td><td>no</td></tr>
<tr><td></td><td>4</td><td>$\wedge\ 2 \cdot |A|+1 < |P|$</td><td>$\wedge\ 2 \cdot |A|+1 < |P|$</td><td>yes</td></tr>
<tr><td></td><td></td><td colspan="3">attacker cannot prevent communication between honest participants</td></tr>
<tr><td>Byzantine agreement</td><td>3.1</td><td colspan="3">$\wedge\ 3 \cdot |A| < |P|$ (Untraceability probabilistic if authentication codes are necessary)</td></tr>
<tr><td></td><td>3.1</td><td colspan="2">$\wedge$ A computationally restricted</td><td>yes</td></tr>
<tr><td></td><td></td><td colspan="3">attacker cannot prevent communication between honest participants</td></tr>
<tr><td>Fail-stop agreement</td><td>3.2</td><td>$\wedge$ comp. restr.</td><td>unconditional</td><td>yes</td></tr>
</table>

Figure 2 Summary

Our solution described in 3.2 is based on **fail-stop Byzantine Agreement**, i.e. Byzantine Agreement with signatures and the additional property that as soon as the attacker forges signatures, all other participants will recognize this. This is realized by a provably secure (one-time) **signature scheme whose forgery by an unexpectedly powerful attacker is provable**.

The method can also be applied to realize **adaptive Byzantine Agreement**, i.e. Byzantine Agreement which can only be disrupted by an attacker who controls at least a third of all participants *and* who can forge signatures (5.1). Other possible applications are payment systems, solving disputes about shared secrets, and signatures which cannot be shown round (5.2 - 5.4).

Acknowledgements

We are pleased to thank Andreas Pfitzmann for a vast amount of tricky ideas, and David Chaum for helpful discussions the attack described in 2.2, which finally yields the protocol described in 2.4, about the signature scheme described in Section 3.2.1, and about the ideas in Chapter 5. We are also grateful to Birgit Baum-Waidner, Manfred Böttger, Axel Burandt, Klaus Echtle, and Tina Johnson for lots of valuable suggestions, and to the German Science Foundation (DFG) for financial support.

References

AlSc_83 Bowen Alpern, Fred B. Schneider: Key exchange Using 'Keyless Cryptography'; Information Processing Letters 16 (Feb. 1983) 79-81.

BeGW_88 Michael Ben-Or, Shafi Goldwasser, Avi Wigderson: Completeness theorems for non-cryptographic fault-tolerant distributed computation; 20th STOC, ACM, New York 1988, 1-10.

BoBo_89 Jurjen Bos, Bert den Boer: Detection of Disrupters in the DC Protocol; Eurocrypt '89 Abstracts; Houthalen, April 1989.

BrCC_88 Gilles Brassard, David Chaum, Claude Crépeau: Minimum Disclosure Proofs of Knowledge; Journal of Computer and System Sciences 37 (1988) 156-189.

Bura_88 Axel Burandt: Informationstheoretisch unverkettbare Beglaubigung von Pseudonymen mit beliebigen Signatursystemen; Studienarbeit, Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, May 1988.

CGMA_85 Benny Chor, Shafi Goldwasser, Silvio Micali, Baruch Awerbuch: Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract); 26th FOCS, IEEE Computer Society, 1985, 383-395.

Cha3_85 David Chaum: The Dining Cryptographers Problem. Unconditional Sender Anonymity; Draft, rec. May 13, 1985.

ChAn_89 David Chaum, Hans van Antwerpen: Undeniable Signatures; Draft, rec. July 21, 1989, announced for Crypto'89.

Chau_88 David Chaum: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability; Journal of Cryptology 1/1 (1988) 65-75.

Chau1_89 David Chaum: The Spymasters Double-Agent Problem – Multiparty Computations Secure Unconditionally from Minorities and Cryptographically from Majorities; rec. July 21, 1989, announced for Crypto '89.

Chau2_89 David Chaum: Private communication; April 12, 1989.

Chau3_89 David Chaum: Private communication; July 21, 1989.

ChCD1_88 David Chaum, Claude Crépeau, Ivan Damgård: Multiparty unconditional secure protocols; 20th STOC, ACM, New York 1988, 11-19.

ChDG_88 David Chaum, Ivan B. Damgård, Jeroen van de Graaf: Multiparty Computations ensuring privacy of each party´s input and correctness of the result; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 87-119.

ChFN_88 David Chaum, Amos Fiat, Moni Naor: Untraceable Electronic Cash; Crypto '88 Abstracts, August 1988.

Damg_88 Ivan Bjerre Damgård: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Heidelberg 1988, 203-216.

DiHe_76 Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory, IT-22/6 (1976) 644-654.

Dole_81 Danny Dolev: The Byzantine Generals Strike Again; Dep. of Comp. Sc., Stanford Univ., Report STAN-CS-81-846, March 1981; appeared in: Journal of Algorithms 3/1 (1982) 14-30.

DoSt_83 Danny Dolev, H. Raymond Strong: Authenticated Algorithms for Byzantine Agreement; SIAM J. Comput. 12/4 (1983) 656-666.

FeMi_88 Paul Feldman, Silvio Micali: Optimal algorithms for byzantine agreement; 20th STOC, ACM, New York 1988, 148-161.

GaHY_88 Zvi Galil, Stuart Haber, Moti Yung: Cryptographic computation: secure fault-tolerant protocols and the public-key model; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 135-155.

GiMS_74 E. N. Gilbert, F. J. Mac Williams, N. J. A. Sloane: Codes which detect deception; The Bell System Technical Journal BSTJ 53/3 (1974) 405-424.

GMW_87 Oded Goldreich, Silvio Micali, Avi Wigderson: How to play any mental game - or - a completeness theorem for protocols with honest majority; 19th STOC, ACM, New York 1987, 218-229.

GMW1_87 Oded Goldreich, Silvio Micali, Avi Wigderson: How to Prove All NP Statements in Zero-Knowledge and a Methodology of Cryptographic Protocol Design; Crypto '86, LNCS 263, Springer-Verlag, Berlin 1987, 171-185.

GoM1_86 Oded Goldreich, Silvio Micali, Avi Wigderson: Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design; 27th FOCS, IEEE Computer Society, 1986, 174-187.

GoMR_88 Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. 17/2 (1988) 281-308.

GrPe_88 Jeroen van de Graaf, René Peralta: A simple and secure way to show the validity of your public key; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 128-134.

Merk_80 Ralph C. Merkle: Protocols for Public Key Cryptosystems; Proceedings of the 1980 Symposium on Security and Privacy, April 1980 Oakland, California, 122-134.

Merk1_82 Ralph C. Merkle: Secrecy, Authentication, and Public Key Systems; UMI Research Press 1982.

Merk_88 Ralph C. Merkle: A digital signature based on a conventional encryption function; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 369-378.

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

Pfi1_85 Andreas Pfitzmann: How to implement ISDNs without user observability - Some remarks; Fakultät für Informatik, Universität Karlsruhe, Interner Bericht 14/85.

Pfit_89 Andreas Pfitzmann: Diensteintegrierende Kommunikationsnetze mit Teilnehmer-überprüfbarem Datenschutz; Univ. Karlsruhe, Fakultät für Informatik, Dissertation, 1989.

Pfit1_89 Birgit Pfitzmann: Für den Unterzeichner unbedingt sichere digitale Signaturen und ihre Anwendung; Diplomarbeit am Institut für Rechnerentwurf und Fehlertoleranz der Universität Karlsruhe, September 1989, in preparation.

Rab1_80 Michael O. Rabin: Probabilistic Algorithms in Finite Fields; SIAM J. Comput. 9/2 (1980) 273-280.

Reis_87 Rüdiger Reischuk: Konsistenz und Fehlertoleranz in Verteilten Systemen - Das Problem der Byzantinischen Generäle; 17. GI Jahrestagung, IFB 156, Springer-Verlag, Berlin 1987, 65-81.

Sim3_88 Gustavus J. Simmons: A Survey of Information Authentication; Proceedings of the IEEE 76/5 (1988) 603-620.

Waid_89 Michael Waidner: Unconditional Sender and Recipient Untraceability in spite of Active Attacks; Universität Karlsruhe 1989; will be published in the Proceedings of Eurocrypt '89, LNCS, Springer-Verlag, Berlin 1989.

WaPf_89 Michael Waidner, Birgit Pfitzmann: Unconditional Sender and Recipient Untraceability in spite of Active Attacks – Some Remarks; Fakultät für Informatik, Universität Karlsruhe, Interner Bericht 5/89, March 1989.

WeCa_81 Mark N. Wegman, J. Lawrence Carter: New Hash Functions and Their Use in Authentication and Set Equality; Journal of Computer and System Sciences 22 (1981) 265-279.