UNIVERSITÄT KARLSRUHE

FAKULTÄT FÜR INFORMATIK

P.O.Box 6980, D 7500 Karlsruhe 1, F. R. Germany

Unconditional Sender and Recipient Untraceability in spite of Active Attacks – Some Remarks

Michael Waidner, Birgit Pfitzmann
Institut für Rechnerentwurf und Fehlertoleranz

Interner Bericht 5/89
März 1989

Table of contents

1 Abstract

In Journal of Cryptology 1/1 (1988) 65-75 (= [Chau_88]) David Chaum describes a technique, the DC-net, to send and receive messages anonymously over an arbitrary network. Section 2 gives a short and slightly generalized description of the DC-net and describes some known reservation techniques.

In [Chau_88] the untraceability of senders and recipients of messages is proved to be unconditional, but this proof implicitly assumes a *reliable* broadcast network, i.e. each message broadcast by an honest participant is received by each other participant without alterations.

Since *unconditional* Byzantine Agreement (i.e. BA in spite of an attacker with unlimited computational power who may control an arbitrary number of participants) is impossible, such a network cannot be realized by cryptographic means. Thus the assumption may be rather unrealistic.

In section 3 it is shown how the sending of a specific participant X can be traced by an active attacker who is able to *manipulate* broadcast and controls the current communication partner of X.

A number of countermeasures, called **fail-stop key generation** schemes, are suggested and it is proved that each of them will realize the desired unconditional untraceability in spite of active attacks.

Section 4 discusses the problem of guaranteeing serviceability while preserving untraceability.

In [Chau_88 sect. 2.5] a protocol for solving this problem is suggested which again depends on the assumption of a reliable broadcast network. It is shown that the protocol is insecure (even on the reliable broadcast assumption): the sender of one randomly selected message can always be identified.

We give several solutions for the problem: Assuming for the attacker on untraceability …

- … reliable broadcast, we can guarantee computationally secure serviceability (sect. 4.3).
- … reliable broadcast and that there is an honest majority of all participants, we can guarantee serviceability on the same assumption (sect. 4.5).
- … that the attacker is not able to prevent Byzantine Agreement, we can guarantee serviceability as secure as the Byzantine Agreement (sect. 4.4.1).
- … that the attacker is not able to prevent the honest participants from communicating (which is considerably less than reliable broadcast), we can guarantee computationally secure serviceability (sect. 4.4.3).

Please notice that the attacker on serviceability is usually weaker than the attacker on untraceability, i.e. there are attackers which may disturb superposed sending without being able to trace messages.

Our fourth solution is based on the problem of digital signatures whose forgery by an unexpectedly powerful attacker is provable. We give a first such (one-time) signature scheme; the forgery of signatures is equivalent to the factoring problem (sect. 4.4.3.1.2).

With such signatures we can realize

- **Fail-stop Byzantine Agreement**, i.e. Byzantine agreement with signatures and the additional property that as soon as the attacker is able to forge signatures all other participants will recognize this (4.4.3.3). This can be used for implementing **fail-stop broadcast**.

- **Adaptive Byzantine Agreement**, i.e. Byzantine agreement which can be disturbed only by an attacker who controls more than a third of all participants *and* who is able to forge signatures (sect. 5).

Some parts of this report will be published in [Waid_89].

2 Unconditional sender untraceability

Section 2.1 describes the basic mechanisms of the DC-net, superposed sending and broadcast, and defines the notation used throughout this paper. Section 2.2 describes an example of an anonymity preserving multi-access protocol for superposed sending, and in section 2.3 some general remarks on sender untraceability schemes are given.

2.1 Superposed sending

Assume that a number of participants want to exchange messages over an arbitrary communication network. A computationally unlimited attacker, who is able to eavesdrop the communication between any two of the participants (e.g. because he collaborates with the network operator) and controls an arbitrary subset of the participants, tries to trace the messages exchanged between the participants to their senders and recipients.

If all messages are delivered to each participant, the attacker is not able to trace the *intended* recipient of a message. Therefore unconditionally reliable broadcast guarantees **unconditional recipient untraceability**.

It is important to notice that in this section as in [Chau_88] attackers are assumed to be unable to manipulate the consistency of broadcast.

Sender untraceability is guaranteed by **superposed sending**, which realizes an anonymous multi-access channel:

Let $P = \{P_1, \ldots, P_n\}$ be the set of all participants and let G be an undirected self-loop free graph with P as nodes. Let $(F,\oplus)$ be a finite abelian group. The set F is called the **alphabet**.

To be able to perform a single sending step, which is called a **round**, each pair of participants P_i, P_j who are directly connected by an edge of G choose a key K_{ij} from F randomly[1]. Let $K_{ji} := K_{ij}$. Participants P_i and P_j keep their common key secret. The graph G is called **key graph**, the tuple K of all keys is called **key combination**.

Each participant P_i chooses a message character M_i from the alphabet F, outputs his *local* sum

$$O_i := M_i \oplus \sum_{\{P_i,P_j\} \in G} \mathrm{sign}(i-j) \bullet K_{ij} \qquad (2.1)$$

[1] In the following the term "X is randomly chosen from a set M" is abbreviated by "$X \in_R M$". This means that X is a uniformly distributed random variable which is independent from "all other variables". What is meant by "all other variables" should always be clear from the context.

and receives as input the *global* sum

$$S := \sum_{j=1}^{n} O_j \tag{2.2}$$

(Fig. 1). As usually the symbolic operation $\pm 1 \bullet K_{ij}$ is defined by $+1 \bullet x := x$ and $-1 \bullet x := -x$.

Because each key is both added and subtracted exactly once, the sum in (2.2) is the sum of all characters M_i. If exactly one character M_i has been chosen unequal to 0, this character is successfully delivered to all participants. Otherwise a (digital) collision occurs which has to be resolved by a multi-access protocol, cf. section 2.2.

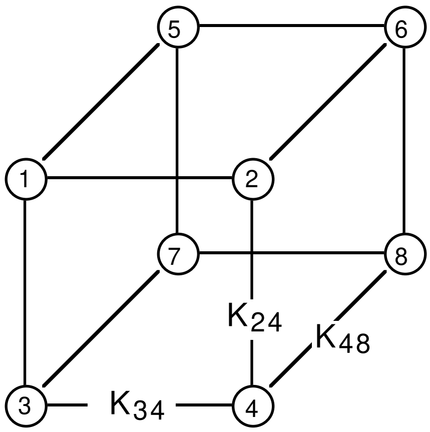


Figure 1 Example of a simple (and incomplete) key graph G. According to eq. (2.1)

$$O_4 = M_4 \oplus K_{24} \oplus K_{34} \oplus -K_{48}$$

According to lemma 2.1, each attacker A with $|A| \leq 2$ can be tolerated.

Superposed sending guarantees **unconditional sender untraceability**. Let A denote the subset of participants controlled by the attacker. If the graph $G \setminus (P \times A)$ is connected, the attacker gets no additional information about the characters M_i besides their sum.

Lemma 2.1 **Superposed sending.** Let A be the subset of participants controlled by the attacker and assume $G \setminus (P \times A)$ to be connected. Let $(O_1,\ldots,O_n) \in F^n$ be the output of a single round.

Then for each vector $(M_1,\ldots,M_n) \in F^n$ which is consistent with the attacker's a priori knowledge about the M_i and which satisfies

$$\sum_{j=1}^{n} O_j = \sum_{j=1}^{n} M_j \tag{2.3}$$

there is the same number of key combinations which satisfy equation (2.1) and which are consistent with the attacker's a priori knowledge about the K_{ij}.

Hence the conditional probability for $(M_1,\ldots,M_n)$ given the output $(O_1,\ldots,O_n)$ (i.e. the a posteriori probability) is equal to the conditional probability for $(M_1,\ldots,M_n)$ given only the sum in (2.3) (i.e. the a priori probability).

This is stated and proved in [Cha3_85, Chau_88] for F = GF(2) by a technique which can easily be applied to any finite field. In [Pfit_89 sect. 2.5.3.1] and in the following, lemma 2.1 is proved for any finite abelian group F. (The general applicability of finite abelian groups was also mentioned in [Pfi1_85].)

Proof. Let $M' := (M'_1,\ldots,M'_n) \in F^n$ be another vector which satisfies (2.3) and which is consistent with the attacker's a priori knowledge about the M_i.

To prove lemma 2.1, a finite sequence $M^0, M^1, \ldots$ of vectors from F^n is defined, which all satisfy eq. (2.3) and which differ in only two components. Let denote $M^k = (M^k_1, \ldots, M^k_n)$.

Let $M^0 := (M_1,\ldots,M_n)$, hence M^0 satisfies eq. (2.3). If $M^k = M'$ then stop. Now assume $M^k \neq M'$.Since both M^k (by induction hypothesis) and M' satisfy eq. (2.3) there are at least two different indices i, j with $M^k_i \neq M'_i$ and $M^k_j \neq M'_j$, and since both M^k and M' are consistent with the attacker's a priori knowledge $P_i, P_j \notin A$. Define

$$\begin{aligned} M^{k+1}{}_i &:= M'_i \\ M^{k+1}{}_j &:= M^k{}_j \oplus M^k{}_i \oplus -M'_i \\ M^{k+1}{}_h &:= M^k{}_h \ \text{ for all } h \notin \{i,j\} \end{aligned} \tag{2.4}$$

Obviously M^{k+1} satisfies (2.3). After at most n-1 steps the sequence stops with $M^k = M'$.

Let K^k be the set of all key combinations which satisfy (2.1) for the vector M^k and which are consistent with the attacker's a priori knowledge. Between each pair K^k, K^{k+1} a bijection ϕ^k is defined. Hence $|K^k| = |K^{k+1}|$ for all k and therefore $|K^0| = |K^{n-1}|$ where $M^{n-1} = M'$.

To define ϕ^k consider the equations (2.4). Let $\Delta := M^{k+1}{}_i \oplus -M^k{}_i$. Then $M^{k+1}{}_i = M^k{}_i \oplus \Delta$ and $M^{k+1}{}_j = M^k{}_j \oplus -\Delta$.

Because of the connectivity of $G \setminus (P \times A)$ there exists a path $(P_i = P_{k_1},\ldots,P_{k_m} = P_j)$ with $P_{k_h} \notin A$ and $(P_{k_h},P_{k_{h+1}}) \in G \setminus (P \times A)$. Let $K \in K^k$. Then $\phi^k(K)$ is defined by changing the keys on this path appropriately:

$$\forall\ h=1,\ldots,m-1:\ \ \phi^k(K)_{k_h k_{h+1}} := K_{k_h k_{h+1}} \oplus -\Delta \bullet \operatorname{sign}(k_h-k_{h+1}),$$
$$\phi^k(K)_{k_{h+1} k_h} := \phi^k(K)_{k_h k_{h+1}}$$

and

$$\forall\ (f,g) \notin \{\ (k_h, k_{h+1}), (k_{h+1}, k_h) \mid h=1,\ldots,m-1\ \}:\ \ \phi^k(K)_{fg} := K_{fg}$$

The construction of ϕ^k is depicted in figure 2.

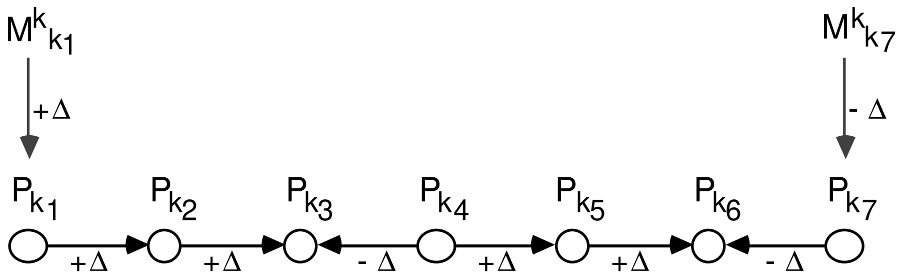
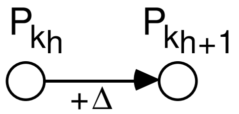


Figure 2 Construction of ϕ^k from a path with m = 7. The vertical arrows indicate the change of $M^k{}_{k_h}$, h=1,7, the horizontal arrows the numerical order of the k_h, the ±Δ the change of $K_{k_h k_{h+1}}$:



$:\Leftrightarrow k_h < k_{h+1}$ and therefore $\phi^k(K)_{k_h k_{h+1}} := K_{k_h k_{h+1}} \oplus \Delta$

Obviously, the local outputs of P_{k_h}, h = 1,…,7, are not changed by ϕ^k.

It can easily be checked that $\phi^k(K)$ satisfies (2.3). Because $\phi^k(K)$ differs from K only in such keys which are unknown to the attacker, $\phi^k(K)$ is necessarily consistent with the attacker's a priori knowledge. Since ϕ^k is simply a translation of the group $F^{|G|}$, the bijectivity of ϕ^k is obvious. □

2.2 Efficient and anonymity preserving multi-access protocols

To use the multi-access channel superposed sending offers it is necessary to regulate the participants' access to the channel by an appropriate, i.e. efficient and anonymity preserving protocol. For an in depth discussion of possible protocols cf. [Pfit_89 sect. 3.1.2].

In the following only three protocols are considered, slotted ALOHA, a bit map reservation technique, and superposed receiving.

The first step for each multi-access protocol is to combine a fixed number c of characters into a **message**. Each message is transmitted in c consecutive rounds, which are called a **slot**.

In the following, rounds are numbered from 1 to a maximum number t_{max}. Parameter t_{max} is necessary only for technical reasons. Usually $ld(|F|) \cdot t_{max}$, i.e. the maximum number of transmitted bits, can be assumed to be very large, e.g. $ld(|F|) \cdot t_{max} = 10^{25}$. Even with a rather unrealistic transmission rate of 10^{15} bps this is sufficient for about 317 years of superposed sending.

The character and output of participant P_i in round t are named M_i^t and O_i^t, respectively, the global sum in round t is named S^t.

The simplest protocol is the well known (slotted) **ALOHA** [Chau_88, Tane_88 sect. 3.2]: If P_i has a message to send he simply does so in the next slot. If another participant has decided to send a message, too, a collision occurs, which is detected by P_i. After waiting a random number of slots, P_i retransmits his message.

Obviously ALOHA preserves anonymity, but wastes the transmission capacity of the network.

To avoid collision of messages a simple **reservation map technique** can be used: a slot of r rounds, the **reservation frame**, is used to reserve the following up to r slots [Pfi1_85 sect. 2.2.2.2].

F is assumed to be the additive group of integers modulo a fixed integer m. For each message P_i plans to send he chooses an index k from {1,…,r} at random and outputs 1 as his k-th character for the reservation frame. The resulting reservation message consists of three classes of characters: 0, indicating an unreserved slot, 1, indicating a reserved slot, and {2,…,m-1}, indicating collisions. Since all message slots with corresponding reservation character ≠1 are of no use, they are skipped, i.e. the reservation frame is followed only by as much message slots as there are successful reservations. Slots with reservation character =1 are used by that participant who has sent a 1 in the corresponding reservation round (fig. 3).

If m is chosen ≥ |P| this scheme avoids any collisions of messages.

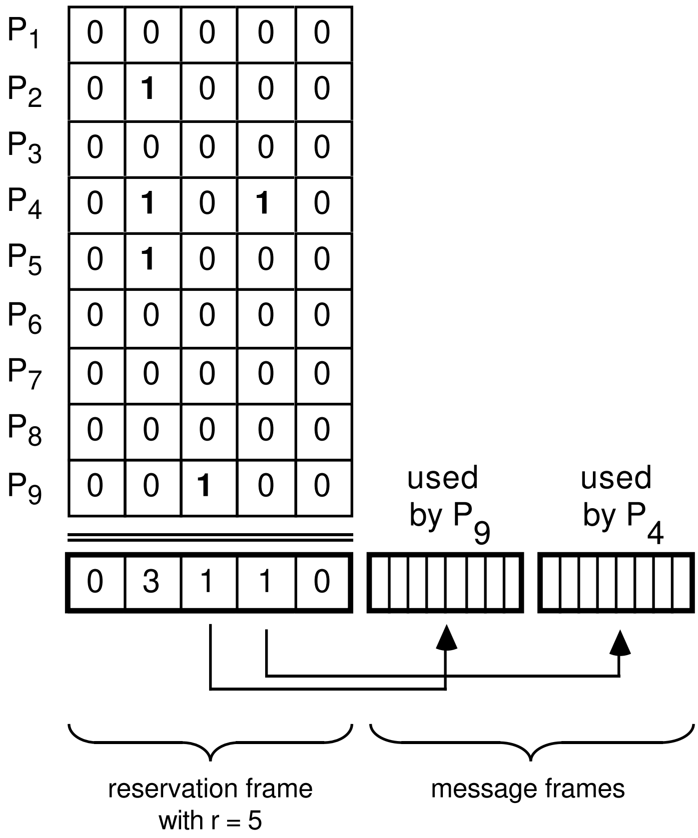


Figure 3 Reservation map technique with 9 participants, a reservation frame of length r = 5, and m ≥ 4. The 1st, 2nd, and 5th slots would be unused and are therefore skipped.

A similar reservation technique is described in [Cha3_85, Chau_88]: instead of using a relatively large group F to enable the detection of multiple collisions, the superposition is done in F = GF(2) and a value r in the order of the square of s_{max}, the maximum number of reservations, is used to make multiple collisions of an odd number of reservations rather unlikely.

Therefore the scheme requires s_{max}^2 additional bits per s_{max} sent messages.

The last multi-access protocol is a collision resolution protocol called **superposed receiving** [Pfit_89 sect. 3.1.2]: it is based on the observation that from knowledge of $M_1 \oplus \ldots \oplus M_n$ and $M_1, \ldots, M_{n-1}$ the last character M_n can easily be derived. (More generally: All M_i can be computed from each set of n linearly independent sums of $M_1, \ldots, M_n$.)

This is used for a recursively defined protocol: Let s_{max} be the maximum number of collided messages, e.g. $s_{max} = n$, and $\{0,1,2,\ldots,M_{max}\} \subset \mathbb{Z}$ the set of all *allowed* message characters. The alphabet F is chosen to be the ring of integers modulo m where m is greater than $s_{max} \bullet M_{max}$. As usual each character $M \in F$ can be interpreted as an integer. A message consists of only two characters: For a participant who has to send a message the first character is always 1 and the second is his message character. For a nonsending participant both are always 0.

Now assume that a new round of the protocol starts and an a priori unknown number of participants have decided to send a message. Let SP denote the set of all sending participants, Σ the sum of their characters M_i modulo m, and s := |SP|. Thus the first slot contains the pair (s, Σ). A number s ≥ 2 indicates a (digital) collision. To resolve it, each participant computes the average message

$$M_A := \left\lfloor \frac{\Sigma}{s} \right\rfloor,$$

which is possible since the modulus m has been chosen so large that Σ is also the sum of the characters in $\mathbb{Z}$. This average is used to deterministically divide the set SP into two disjoint subsets SP_1 and SP_2: SP_1 consists of all participants $P_i \in SP$ with $M_i \leq M_A$, SP_2 consists of all other sending participants:

$$SP_1 := \{ P_i \in SP \mid M_i \leq M_A \}$$

$$SP_2 := \{ P_i \in SP \mid M_i > M_A \}$$

For i = 1,2 define s_i and Σ_i in analogy to s and Σ. All participants $P_i \in SP_1$ immediately repeat their messages $(1,M_i)$ in the next slot, hence each user *receives* the pair (s_1, Σ_1) and can *compute* the pair $(s_2, \Sigma_2) = (s \oplus -s_1, \Sigma \oplus -\Sigma_1)$.

Given the rare case $s_2 = 0$, the protocol terminates after the second slot: each participant $P_i \in SP$ has sent the *same* character $M_i = M_A$. Otherwise, i.e. $s_2 \neq 0$, the sets SP_1 and SP_2 are both nonempty and the collision resolution procedure is recursively applied to (s_i, Σ_i), i = 1,2.

To resolve a collision of s messages the protocol deterministically needs at most s slots, which is optimal. (For a performace evaluation of superposed sending cf. [Marc_88].)

Figure 4 gives an example for the resolution of a collision of s = 5 messages.

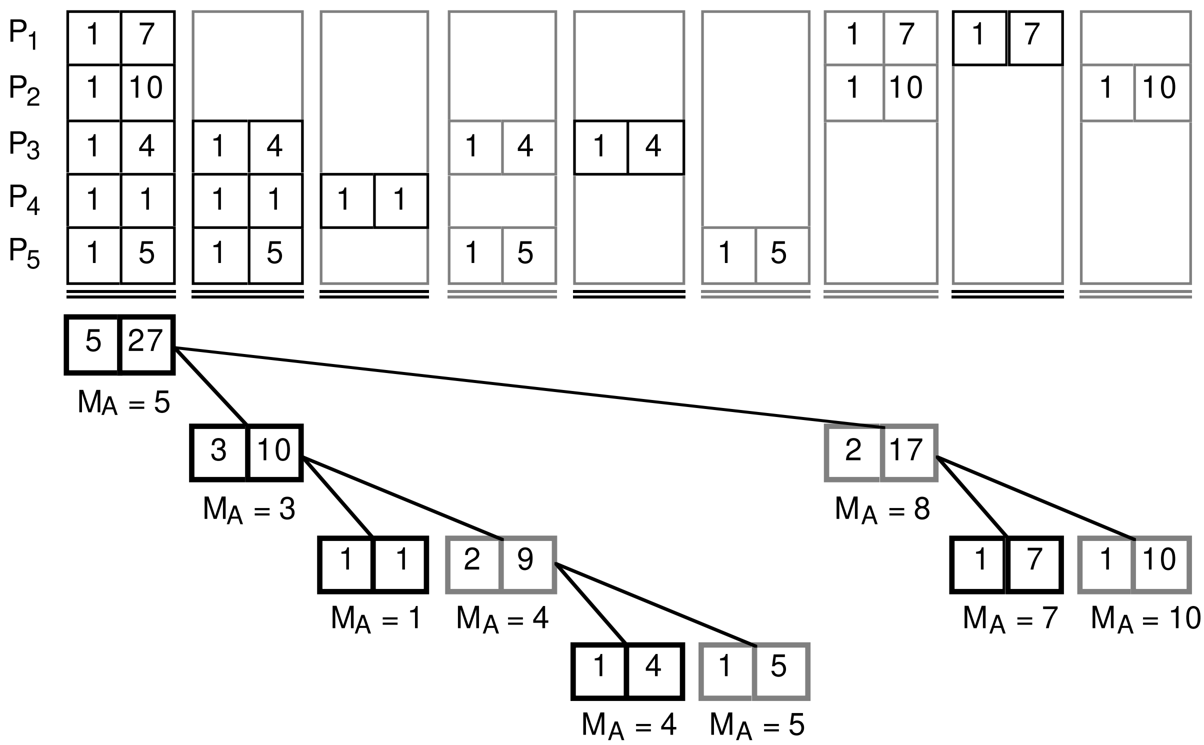


Figure 4 Collision resolution using superposed receiving. The black boxes indicate really sent messages, the hatched boxes are only computed. Using the tree structure all 5 messages can be computed from the 5 really sent messages.

To allow the sending of long messages either the alphabet F could be made large enough to represent a long message by a single character or superposed receiving could be used as a reservation technique (**reservation by superposed receiving**): each participant willing to send a message chooses at random a reservation message $(1,RM_j)$ and sends it in the next possible reservation slot. The collision of all s reservation messages is resolved by superposed receiving, after which each P_i sorts the received reservation characters RM_j according to their numerical values (in fact, P_i receives the reservation characters in their numerical order!). With high probability all RM_j are different, therefore the order of the RM_j naturally defines an order of all reserving participants, according to which each P_j sends his real message in the appropriate one of the next s slots. If some RM_j are equal, they don't lead to any reservation (i.e. although superposed receiving is a deterministic collision resolution scheme, the resulting reservation scheme is not completely deterministic, but only successful with very high probability.)

2.3 Some remarks on sender untraceability schemes

Given the very strong assumption of an unlimited attacker (i.e. there may be an arbitrary number of attackers $|A| < |P|$, there are no computational restrictions) the fundamental restrictions of superposed sending as far as performance and reliability are concerned are a consequence of its sender untraceability: In order to make the physical behaviour of a participant meaningless it is necessary that a participant P_i who is willing to send a character M_i

- does this in an encrypted way,
- each other participant P_j outputs a character, too, and
- the attacker is not able to learn anything about M_i before knowing all the outputs.

Because the attacker is assumed to be an insider it follows from the last fact that the result of such a single sending step cannot contain more information than the last of the participant's output does. Therefore any unconditional sender untraceability scheme realizes a multi-access channel and superposed sending offers the best possible channel capacity as far as only a single round is concerned.

To guarantee the unconditional sender untraceability, the global output of the realized multi-access channel has to depend on each participant's output, therefore any unconditional sender untraceability scheme can be untraceably disturbed by each participant.

As far as I know, superposed sending is the only *unconditional* sender untraceability scheme which withstands an unlimited attacker.

There are two other untraceability schemes known from literature, the MIX-net [Chau_81] and the concept of physical unobservability [Pfit_84]. Both can only withstand weaker attackers than superposed sending. The first is based on the use of a public-key cryptosystem and the existence of a number of network stations, called MIXes, from which at least one has to be trustworthy. The second assumes that the attacker only controls a very small number of participants.

To reduce the tremendous amount of randomly chosen keys for superposed sending which have to be exchanged by the participants, one can use keys which are generated by pseudorandom bitgenerators (PRBG). If the used PRBG is cryptographically strong, i.e. if distinguishing the PRBG from a true random source in random polynomial time is provably equivalent to solving a (hopefully) hard problem [VaVa_85], tracing becomes equivalent to this hard problem, too, but the *unconditional* sender untraceability is lost.

In [BeGW_88, ChCD1_88] very general techniques for information theoretically secure fault tolerant distributed computations are described. In general these techniques can be used for implementing a sender and recipient untraceability scheme. Since they are based on the well known reliable broadcast problem (cf. section 3.1) they can withstand only attackers with $3 \cdot |A| < |P|$ and are therefore not further considered here. Also, an untraceability scheme based on such a general technique would be far more expensive than superposed sending together with any of the techniques described in section 3.

On the other hand these techniques ensure serviceability: If instead of a channel with collisions like that of superposed sending a channel which e.g. transmits *all* submitted characters $M_1, \ldots, M_n$ in their numerical order is realized, it would be guaranteed that no participant can disrupt the sending of the M_i. The input phase of such a network (i.e. the subprotocol by which all participants share their secrets M_i among all others) can be made attacker and fault tolerant by standard techniques without loss of untraceability. Hence most of the reliability problems of superposed sending (discussed in the section 4) wouldn't be posed in such a network.

Because of the growing importance of public telecommunication networks it seems necessary to look for efficient implementations of untraceability schemes resulting in networks without user observability. For more details about the motivation and the more practical aspects of this task cf. [Pfi1_85, PfWa_86, PfPW_88, Pfit_89].

3 Active attacks on untraceability

The power of an *active* attack is based on a very simple observation: for services using two-way communication it is impossible to realize unconditional sender untraceability without unconditional recipient untraceability and vice versa.

To see this, assume that one of the participants controlled by the attacker, say P_a, communicates with some honest participant X and that X will answer a message M by sending a message M'. If the attacker is able to identify the *sender* of M' he can identify the *recipient* of M and vice versa. If the attacker doesn't control P_a the same is true for light traffic; then the attacker can identify both communication partners.

In general if sending and receiving is correlated (which is usually the case) the attacker can always learn something about recipients from identifying senders and vice versa.

If active attacks are possible, superposed sending doesn't guarantee recipient untraceability and therefore it doesn't guarantee sender untraceability:

Let I_i (I_i^t) be the input character which participant P_i receives (in round t) and which should always be equal to the global sum S (S^t).

Assume that the attacker is able to deliver an arbitrary character I^*_i to each participant P_i instead of the correct character I_i. Further assume that participant P_a, who is controlled by the attacker, communicates with the honest participant X according to some protocol. P_a knows that X will always answer to a received message M within a given time by sending a message M'.

If the attacker delivers message M consecutively only to a single participant and a meaningless message to all others, he can always identify X by checking when he receives M' or not. Instead of delivering M only to a single participant he can deliver it to a subset of the participants. By successively partitioning the participants he can identify X in log(n) rounds, provided that the protocol between X and P_a consists of at least log(n) interactions (on the average $\frac{\log(n)}{2}$ interactions would suffice).

If it were *guaranteed* that in all rounds $t=1,\ldots,t_{max}$ each participant not controlled by the attacker receives the same input character, then superposed sending would guarantee unconditional sender and recipient untraceability in the presence of arbitrary active attacks. Such a network is called a **DC^+ -net**.

For an a priori given number t_{max} of rounds this is the well known problem of **reliable broadcast**. Instead of using a fixed t_{max} one can also try to limit t_{max} adaptively: if in round t two honest participants receive different characters then t_{max} is set to t; this is called **fail-stop broadcast** here.

In the following, two different implementations of superposed sending are considered: the **centralized** and the **distributed implementation**. In a centralized implementation superposition of the local sums is done by a central station, e.g. the centre of a star network, which delivers the global sum to all participants. In a distributed implementation each participant receives the local sum of each other, e.g. via a broadcast channel, and computes the global sum locally for himself.

In a centralized implementation based on a star network the attack described above is possible for the centre without any manipulation of communication lines.

3.1 Reliable broadcast

Reliable broadcast is defined by the following two properties [PeSL_80]: in each round t

i. every two honest participants P_i and P_j receive the same character, i.e. $I_i^t = I_j^t$, and

ii. if the "sender" X is honest, then each honest participant receives the character sent by X.

In a centralized implementation only the network centre has the function of a "sender", in a distributed implementation each participant.

Some types of networks, e.g. satellite networks, offer reliable broadcast without any additional protocol, but because of their bandwidth limitations they are not very usual in two-way telecommunication. Also the DC-network is meant to be usable with a variety of underlying communication networks, e.g. rings, therefore a cryptographic solution should be preferred to a physical one.

The problem of achieving reliable broadcast on a network which does not provide it automatically is also known as the Byzantine Generals problem, its solution by protocols as **Byzantine Agreement** [PeSL_80, LaSP_82].

It has been proved that *information-theoretically* secure protocols for reliable broadcast exist iff the number of honest participants is greater than twice the number of dishonest participants, i.e. $|P| > 3 \cdot |A|$, and the attacker is not able to prevent the communication between honest participants [LaSP_82]. All protocols for information-theoretically secure reliable broadcast implicitly make use of perfect authentication codes [GiMS_74, Sim3_88] and therefore require a large number of additional secret keys exchanged by the participants. Based on the existence of secure signatures there are reliable broadcast protocols for arbitrary numbers $|A| < |P|$ [LaSP_82]. They are usually more efficient than the information-theoretically secure solutions but cannot guarantee unconditional recipient untraceability due to the impossibility of unconditionally secure digital signatures.

Because of its severe limitation $|P| > 3 \cdot |A|$ reliable broadcast does not seem to be a useful technique for the desired unconditional recipient untraceability and is therefore not further considered in section 3. We will return to it in section 4.4, when we have additional requirements on serviceability.

Fail-stop broadcast combines both advantages: it can be implemented in a more efficient way than reliable broadcast and it is unconditionally secure in spite of arbitrary attackers.

3.2 Fail-stop broadcast

The goal of fail-stop broadcast is to stop message transmission as soon as two honest participants receive different input characters.

If such a difference is detected by an honest participant P_i the fail-stop can easily be performed: P_i simply disturbs the superposed sending in the subsequent rounds by choosing his outputs randomly from F instead of following eq. (2.1). Then the global sums of all subsequent rounds are independent of the message characters.

In section 3.2.1 the most obvious but inefficient implementation of this idea by a comparison protocol is discussed.

In section 3.2.2 fail-stop key generation schemes are described: they generate keys for superposed sending dependent on the received input characters and ensure that two participants who have received different input characters will use completely independent keys (at least with high probability) and thus will stop message transmission.

It is shown that the most efficient key generation scheme (sect. 3.2.2.2) does not affect the performance and reliability characteristics of pure superposed sending.

3.2.1 Comparison of input characters

To detect a difference the participants can explicitly compare their input characters by an additional protocol: After each round of superposed sending each participant P_i sends his input character I_i to all participants P_j with $j > i$. If an honest participant P_j receives an input character unequal to I_j from another participant P_i or if he receives nothing from a P_i with $i < j$, he will disturb superposed sending in all subsequent rounds.

Such test phases are well known from Byzantine Agreement protocols.

To make the tests dependable, the communication between P_i and P_j should be protected by a perfect authentication scheme [GiMS_74, Sim3_88], i.e. a scheme which allows the attacker to successfully forge a message with probability at most $1 / \sqrt{|F|}$, if F is used as key space. An additional message and a secret key are therefore necessary for each test.

The necessary number of tests can be determined according to the attacker's assumed power: define G^* to be an undirected graph whose nodes are the participants. Two participants P_i and P_j are directly connected in G^* iff P_i and P_j compare their input characters. In analogy to superposed sending, the following lemma 3.1 holds:

Lemma 3.1 Let A be the subset of participants controlled by the attacker and assume $G^* \setminus (P \times A)$ to be connected.

If two honest participants P_i and P_j receive different input characters I_i, I_j, then there exists a pair of honest participants $P_{i'}$ and $P_{j'}$ who are directly connected in G^* and also receive different input characters.

Hence either $P_{i'}$ or $P_{j'}$ detects the difference and disturbs superposed sending.

Proof. Because of the connectivity of $G^* \setminus (P \times A)$ there exists a path $(P_i = P_{k_1},\ldots,P_{k_m} = P_j)$ with $P_{k_z} \notin A$ and $(P_{k_z},P_{k_{z+1}}) \in G^* \setminus (P \times A)$. It is assumed that $I_i \neq I_j$, hence there exists an index z such that $I_{k_z} \neq I_{k_{z+1}}$. Choose $(i', j') = (k_z,k_{z+1})$. □

Obviously the connectivity of $G^* \setminus (P \times A)$ is a necessary condition.

The scheme requires $|G^*|$ additional messages in each round, which is usually in the order of $O(n^2)$. If $G = G^*$, and if it is assumed that for each test message the authentication scheme requires a key chosen from F, the number of privately exchanged keys is increased by a factor of two in comparison with pure superposed sending.

In a physical broadcast environment the number of test messages can be reduced to O(n) broadcasted messages by using a digital signature scheme [DiHe_76, GoMR_88] instead of an authentication scheme. But this results in an only computationally secure scheme.

3.2.2 Message dependent key generation

3.2.2.1 Deterministic fail-stop key generation

A more efficient realization of fail-stop broadcast is obtained by combining the tasks of detection of differences and stopping the network: if the keys K_{ij} and K_{ji} used for superposed sending depend completely (but not exclusively) on the characters received by P_i and P_j, then a difference between I_i and I_j will automatically disturb superposed sending, thereby stopping message transmission.

Define $\delta_{ij}^t := K_{ij}^t \oplus -K_{ji}^t$ and $\varepsilon_{ij}^t := I_i^t \oplus -I_j^t$ for all i, j, t. A key generation scheme for superposed sending is required which guarantees for all P_i and P_j directly connected in G:

SS *Superposed sending:* If for all rounds s = 1,…,t-1 the equation $I_i^s = I_j^s$ holds, then the keys K_{ij}^t and K_{ji}^t for round t are *equal* and randomly selected from F. More formally:

$$[\forall\, s \in \{1,\ldots,t\text{-}1\}\colon\ \varepsilon_{ij}^s = 0] \Rightarrow K_{ij}^t \in_R F \text{ and } \delta_{ij}^t = 0$$

Then superposed sending works as usual.

FS *Fail-stop:* If there exists an index s < t with $I_i^s \neq I_j^s$, then the keys K_{ij}^t and K_{ji}^t for round t are *independently* and randomly selected from F. More formally:

$$[\exists\, s \in \{1,\ldots,t\text{-}1\}\colon\ \varepsilon_{ij}^s \neq 0] \Rightarrow K_{ij}^t \in_R F \text{ and } \delta_{ij}^t \in_R F$$

Superposed sending is disturbed by any such pair, i.e. the global sum is independent of the sent message characters. Because of the connectivity of $G \setminus (A \times P)$ this realizes the fail-stop property according to lemma 3.1 (with $G=G^*$).

In the rest of section 3.2.2 an arbitrary but fixed key pair (K_{ij},K_{ji}) with $P_i \notin A$ and $P_j \notin A$ is considered. Therefore indices i, j are often omitted.

The most powerful attacker is assumed: he is able to observe the values of K_{ij}^t and K_{ji}^t for each round t directly and he can deliver arbitrary input characters I_i^t and I_j^t to P_i and P_j. Participants P_i and P_j are assumed to by unsynchronized, hence the attacker can wait for K_{ij}^{t+1} before he delivers I_j^t to P_j.

Let $(F,+,\bullet)$ be a finite field and let $a^1, a^2, \ldots, a^{t_{max}}$ and $b^1, b^2, \ldots, b^{t_{max}-1}$ be two sequences whose elements are randomly selected from F and privately exchanged by P_i and P_j. Define for $t = 1,\ldots,t_{max}$

$$K_{ij}{}^{t} := a^{t} + \sum_{k=1}^{t-1} b^{t-k} \bullet I_i{}^{k}$$
$$K_{ji}{}^{t} := a^{t} + \sum_{k=1}^{t-1} b^{t-k} \bullet I_j{}^{k} \tag{3.1}$$

Lemma 3.2 The key generation scheme defined by equation (3.1) satisfies the two conditions SS and FS formulated above.

Proof. Since $a^t \in_R F$, and since $\sum := K_{ij}{}^t - a^t$ is independent of a^t, $K_{ij}{}^t \in_R F$.

Assume $\varepsilon_{ij}{}^s = 0$ for all $s < t$. Then obviously $\delta_{ij}{}^t = 0$ and condition SS is satisfied.

Now assume that s is the first round with $\varepsilon_{ij}{}^s \neq 0$. For simplicity let $\varepsilon^u := \varepsilon_{ij}{}^u$ and $\delta^u := \delta_{ij}{}^u$. The differences δ^u are formed according to the following system of linear equations:

$$\delta^u = 0 \text{ for } u=1,\ldots,s$$

$$\begin{pmatrix} \delta^{s+1} \\ \delta^{s+2} \\ \ldots \\ \delta^{t-1} \\ \delta^{t} \end{pmatrix} = \begin{pmatrix} \varepsilon^{s} & 0 & \ldots & 0 & 0 \\ \varepsilon^{s+1} & \varepsilon^{s} & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \varepsilon^{t-2} & \varepsilon^{t-3} & \ldots & \varepsilon^{s} & 0 \\ \varepsilon^{t-1} & \varepsilon^{t-2} & \ldots & \varepsilon^{s+1} & \varepsilon^{s} \end{pmatrix} \bullet \begin{pmatrix} b^{1} \\ b^{2} \\ \ldots \\ b^{t-s-1} \\ b^{t-s} \end{pmatrix}$$

Since $\varepsilon^s \neq 0$, the matrix is regular and defines a bijective mapping. Since all $b^u \in_R F$, all δ^u are uniformly and independently distributed in F. The independence of all $K_{ij}{}^1,\ldots,K_{ij}{}^t$ and $\delta^{s+1},\ldots,\delta^t$ follows from the independence of all $a^1,\ldots,a^t$ and $\delta^{s+1},\ldots,\delta^t$. □

The additional expenditure of this key generation scheme is given by

- the $2 \bullet t_{max} - 1$ privately exchanged keys a^t, b^t for each pair P_i, P_j directly connected in G (instead of only t_{max} for pure superposed sending),
- the storage of all t_{max}-1 received input characters, and
- the (t-1) field additions and multiplications for computing the key for round t.

From the last fact it follows that the scheme requires an average of $\frac{t_{max}}{2}$ field additions and multiplications per round. Hence the scheme seems not to be very practical.

Given the assumption that there is no additional communication between P_i and P_j about their current states the scheme is *optimal* with respect to the number of exchanged keys and additional storage requirements.

Lemma 3.3 The key generation scheme defined by equation (3.1) is optimal with respect to the number of exchanged keys and additional storage requirements, i.e. each key generation scheme which *deterministically* satisfies conditions SS and FS requires at least

- the storage of all t_{max} - 1 received input characters and
- $2 \bullet t_{max}$ - 1 privately exchanged keys.

Proof. The first limit is obvious: the scheme has to distinguish between all possible sequences of t_{max} input characters, hence all input characters have to be stored.

For proving the second limit let Z be the secret key shared by P_i and P_j and used for generating the keys K_{ij}^t, K_{ji}^t, $t = 1,\ldots,t_{max}$. Let $H(I_i)$, $H(I_j)$, $H(K_{ij}^1)$, $H(K_{ij}^{(2,t_{max})})$, $H(K_{ji}^{(2,t_{max})})$, H(Z) be the entropy of the random variables $I_i = (I_i^1,\ldots,I_i^{t_{max}})$, $I_j = (I_j^1,\ldots,I_j^{t_{max}})$, K_{ij}^1 $(=K_{ji}^1)$, $K_{ij}^{(2,t_{max})} = (K_{ij}^2,\ldots,K_{ij}^{t_{max}})$, $K_{ji}^{(2,t_{max})} = (K_{ji}^2,\ldots,K_{ji}^{t_{max}})$, and Z, respectively [Gall_68, sect. 2].

By applying standard rules of information theory

$$H(K_{ij}^1 K_{ij}^{(2,t_{max})} K_{ji}^{(2,t_{max})} \mid I_i I_j) \leq H(Z K_{ij}^1 K_{ij}^{(2,t_{max})} K_{ji}^{(2,t_{max})} \mid I_i I_j)$$
$$= H(Z \mid I_i I_j) + H(K_{ij}^1 K_{ij}^{(2,t_{max})} K_{ji}^{(2,t_{max})} \mid Z I_i I_j)$$

Since Z is chosen independently from the attacker's input characters $H(Z \mid I_i I_j) = H(Z)$, and since the keys are completely determined by Z and I_i, I_j, $H(K_{ij}^1 K_{ij}^{(2,t_{max})} K_{ji}^{(2,t_{max})} \mid Z I_i I_j) = 0$.

Hence it follows

$$H(Z) \geq H(K_{ij}^1 K_{ij}^{(2,t_{max})} K_{ji}^{(2,t_{max})} \mid I_i I_j)$$

Since only a lower bound is proved, it can be assumed that the attacker chooses I_i^1 and I_j^1 differently. Then the keys K_{ij}^1, and K_{ij}^t, K_{ji}^s for $t, s = 2,\ldots,t_{max}$ are independently chosen, i.e.

$$H(K_{ij}^1 K_{ij}^{(2,t_{max})} K_{ji}^{(2,t_{max})} \mid I_i I_j) = H(K_{ij}^1 K_{ij}^{(2,t_{max})} K_{ji}^{(2,t_{max})})$$
$$= H(K_{ij}^1) + H(K_{ij}^{(2,t_{max})}) + H(K_{ji}^{(2,t_{max})})$$

Hence

$$H(Z) \geq H(K_{ij}^1) + H(K_{ij}^{(2,t_{max})}) + H(K_{ji}^{(2,t_{max})})$$

i.e. Z must consist of at least $1 + (t_{max}-1) + (t_{max}-1) = 2 \bullet t_{max} - 1$ keys. □

3.2.2.2 Probabilistic fail-stop key generation

To get a more efficient key generation scheme it seems necessary to switch to a probabilistic version of FS: For a given fail-stop mechanism let $Prob_A$ be the attacker's **probability of success**. The attacker is successful if in spite of choosing $I_i^s \neq I_j^s$ for a $s < t_{max}$ there exists an index t, $s < t \leq t_{max}$, such that the global sum S^t and the message characters M_i^t, $i=1,\ldots,n$, are *not* independent.

For each $d \in \mathbb{N}$ define

FS_d If two honest participants receive two different input characters in round t they will disturb superposed sending for the following d rounds.

The maximum number d for which FS_d is satisfied is a random variable with probability distribution Prob(d).

Let $a^1, a^2, \ldots, a^{t_{max}}, b^3, b^4, \ldots, b^{t_{max}}$, e be randomly and privately selected elements of the finite field F. Let $b^1 = b^2 = 0$ and let $K_{ij}^0 = K_{ji}^0 = 0$ and $I_i^0 = I_j^0 = 0$. Then define for $t = 1,\ldots,t_{max}$

$$K_{ij}^t := a^t + b^t \bullet K_{ij}^{t-1} + e \bullet I_i^{t-1}$$
$$K_{ji}^t := a^t + b^t \bullet K_{ji}^{t-1} + e \bullet I_j^{t-1} \qquad (3.2)$$

Lemma 3.4 The key generation scheme defined by equation (3.2) satisfies condition SS. The maximum number d for which FS_d is satisfied is a geometrically distributed random variable:

$$\mathrm{Prob}(d) = \frac{1}{|F|} \bullet \left(1 - \frac{1}{|F|}\right)^{d-1}$$

The attacker's probability of success is

$$\mathrm{Prob}_A \leq 1 - \left(1 - \frac{1}{|F|}\right)^{t_{max}}$$

Proof. Since $a^t \in_R F$, and since $\sum := K_{ij}^t - a^t$ is independent of a^t, $K_{ij}^t \in_R F$.

Assume $\varepsilon_{ij}^s = 0$ for all $s < t$. Then obviously $\delta_{ij}^t = 0$ and condition SS is satisfied.

Now assume that s is the first round with $\varepsilon_{ij}^s \neq 0$. For simplicity let $\varepsilon^v := \varepsilon_{ij}^v$ and $\delta^v := \delta_{ij}^v$.

In the next round $\delta^{s+1} = e \bullet \varepsilon^s$. Since $\delta^v = 0$ for all $v \leq s$ the attacker has no information about the actual value of e before round s+1. By assumption $\varepsilon^s \neq 0$, hence δ^{s+1} is uniformly distributed in F.

Now consider the rounds $s + u + 1$ with $u \geq 1$. If $\delta^{s+u} = 0$, then $\delta^{s+u+1} = e \bullet \varepsilon^{s+u}$. From round s+1 the attacker knows the value of e, hence δ^{s+u+1} is *not* independently distributed in F. If $\delta^{s+u} \neq 0$, then $\delta^{s+u+1} = b^{s+u+1} \bullet \delta^{s+u} + e \bullet \varepsilon^{s+u}$. Since b^{s+u+1} is uniformly distributed in F, δ^{s+u+1} is uniformly distributed, too, and since b^{s+u+1} is only used in that round, δ^{s+u+1} is independent of all other δ's.

Therefore the actual value of d is given by the least value $d \geq 1$ for which $\delta^{s+d} = 0$. Since δ^{s+1} is uniformly distributed,

$$\mathrm{Prob}(\delta^{s+1} \neq 0) = 1 - \frac{1}{|F|}$$

and since for $\delta^{s+d} \neq 0$, δ^{s+d+1} is uniformly distributed,

$$\mathrm{Prob}(\delta^{s+d+1} \neq 0 \mid \delta^{s+d} \neq 0) = 1 - \frac{1}{|F|}$$

From this it follows

$$\mathrm{Prob}(d) = \frac{1}{|F|} \bullet \left(1 - \frac{1}{|F|}\right)^{d-1}$$

The independence of all $K_{ij}^1,\ldots,K_{ij}^t$ and $\delta^{s+1},\ldots,\delta^d$ follows from the independence of all $a^1,\ldots,a^t$ and $\delta^{s+1},\ldots,\delta^d$.

The probability of success is simply the probability that $s + d \leq t_{max}$:

$$\mathrm{Prob}_A = \mathrm{Prob}(d \leq t_{max} - s)$$

Since $s \geq 0$,

$$\mathrm{Prob}_A \leq \mathrm{Prob}(d \leq t_{max}) = 1 - \mathrm{Prob}(d > t_{max}) = 1 - \left(1 - \frac{1}{|F|}\right)^{t_{max}}$$

□

Since d is geometrically distributed the average value of d is |F| [Triv_82 p. 579]. Hence |F| must be chosen considerably larger than t_{max}.

Corollary. Assume the key generation scheme of eq. (3.2). Then

$$\mathrm{Prob}_A \leq 1 - \left(\frac{1}{4}\right)^{t_{max} / |F|}$$

Proof. From lemma 3.4 it follows

$$Prob_A \leq 1 - (1-\frac{1}{|F|})^{t_{max}} = 1 - (1-\frac{1}{|F|})^{|F| \bullet t_{max} / |F|}$$

The sequence $(1 - \frac{1}{x})^x$ increases monotonously. Since $|F| \geq 2$

$$Prob_A \leq 1 - (\frac{1}{4})^{t_{max} / |F|}$$

□

Obviously with a decreasing value of $t_{max} / |F|$ the probability $Prob_A$ vanishes. From the corollary it follows for each $0 \leq L < 1$

$$\frac{t_{max}}{|F|} \leq \frac{1}{2} \bullet ld(\frac{1}{1-L}) \;\Rightarrow\; Prob_A \leq L$$

E.g. for $L = 10^{-9}$

$$\frac{t_{max}}{|F|} \leq 7 \bullet 10^{-10}$$

is sufficient, which is satisfied e.g. by $|F| = 2^{108}$ and $t_{max} = 10^{23}$. These values allow the transmission of

$$t_{max} \bullet ld(|F|) = 10^{23} \bullet 108 \text{ bit} \approx 10^{25} \text{ bit}$$

For a transmission speed of 10^{15} bit/s (which is far beyond today's technology) this would be sufficient for about 317 years.

The key generation of eq. (3.2) requires as many privately exchanged keys as the scheme defined by eq. (3.1), i.e. $2 \bullet t_{max} - 1$.

To evaluate eq. (3.2) for round t it is only necessary to store the last key, K_{ij}^{t-1} (in contrast to the last t-1 keys for eq. (3.1)) and to perform 2 field additions and multiplications. In contrast to the scheme of eq. (3.1), only large fields are suitable.

3.2.2.3 Combination of key generation and explicit tests

If the multi-access protocol guarantees that for some slots only one participant is allowed to choose a nonzero message, this participant can test the network:

Assume that superposed sending is stopped after a broadcast inconsistency by one of the key generation schemes described above, i.e. the global sums are randomly distributed. Then each participant P_i who is allowed to use a slot exclusively and sends a message randomly selected from F^c will receive a wrong message with probability $1 - |F|^{-c}$. Thus he detects the disturbance with the same probability and can explicitly stop superposed sending by choosing his following output characters randomly from F instead of according to eq. (2.1).

If it is guaranteed that each participant sends a test message within a fixed number s of slots and if there are at least two honest participants, this makes it unnecessary to consider more than the last $(s-1) \bullet c$ input characters for key generation: after s-1 slots superposed sending will be explicitly disturbed with high probability by some honest participant who received a disturbed test message instead of that one he sent.

The required fairness of the multi-access protocol can deterministically be satisfied by superposed receiving and in a probabilistic sense e.g. by reservation by superposed receiving (sect. 2.1). If e.g.

each participant reserves exactly one test message and at most one real message in each reservation phase, each participant tests the network within $s = 4 \bullet n$ slots.

Obviously this fairness can only be guaranteed if all participants behave fair, i.e. each unfair (and therefore dishonest) participant can prevent some honest participants from successfully doing their required reservation. Therefore each honest participant who cannot send a message within s slots should disturb superposed sending.

The additional rules don't help the attacker: Assume that an honest participant P_i detects a disturbance, i.e. $I_i^t \neq M_i^t$, and stops sending. Nevertheless the attacker is not able to observe the sending of P_i.

If the disturbance detected by P_i was a consequence of a previous broadcast inconsistency the sending was stopped anyway, thus there is nothing to show. Otherwise and if all honest participants receive the same input character, the unobservability of P_i follows from lemma 2.1, and if the attacker manipulates the broadcast property for round t, the sending is stopped by the key generation scheme anyway, independent of P_i's test.

The proper modifications of the key generation schemes will be discussed in the following two sections.

The advantages and disadvantages of the combination are the same in both schemes:

- For key generation the parameter t_{max} is replaced by $(s-1) \bullet c$, which decreases the number of additional secret keys from t_{max} to $(s-1) \bullet c$, and for deterministic key generation the computation complexity from $O(t_{max}^2)$ to $O(s^2 \bullet c^2)$ operations and from $O(t_{max})$ to $O(s \bullet c)$ required storage.
- Some honest participants may be forced to send meaningless test messages, thus the throughput of the DC-net is decreased. The number of additional test messages depends on the participants' sending rates.

3.2.2.3.1 Combination of deterministic key generation and explicit tests

Assume that the deterministic scheme of eq. (3.1) is used in combination with explicit tests.

If round u is the first disturbed round, the attacker has no information about the privately exchanged keys b^v, $v = 1,\ldots,u$. After round $u + (s-1) \bullet c$ the DC^+-net will be disturbed with high probability by at least one honest participant who has detected the disturbance. Hence instead of $t_{max} - 1$ additional keys at most $(s-1) \bullet c$ are really necessary:

$$K_{ij}^t := a^t + \sum_{k=t-(s-1)\bullet c}^{t-1} b^{t-k} \bullet I_i^k$$
$$K_{ji}^t := a^t + \sum_{k=t-(s-1)\bullet c}^{t-1} b^{t-k} \bullet I_j^k \qquad (3.3)$$

Lemma 3.5 The key generation scheme defined by equation (3.3) satisfies condition SS. Together with the additional rules for testing and disturbing it ensures the fail-stop property in a probabilistic sense: Let h be the number of honest participants, $h \geq 2$. Then

$$\mathrm{Prob}_A \leq \frac{1}{|F|^{c \bullet (h-1)}}$$

Proof. Since $a^t \in_R F$, and since $\sum := K_{ij}{}^t - a^t$ is independent of a^t, $K_{ij}{}^t$ is uniformly distributed in F. Assume $\varepsilon_{ij}{}^u = 0$ for all $u < t$. Then obviously $\delta_{ij}{}^t = 0$ and condition SS is satisfied.

Now assume that u is the first round with $\varepsilon_{ij}{}^u \neq 0$. According to lemma 3.3 (with $t_{max} = (s\text{-}1)\bullet c\text{-}1$) the global sums of the following (s-1)•c-1 rounds are all randomly chosen from F. Since it is assumed that during the s slots each participant tests the network, the only chance of the attacker is that during the first s-1 slots none of the at least h-1 honest participants detects the disturbance. The probability that a single test doesn't detect a disturbance is $|F|^{-c}$, hence the attacker's probability is less than $|F|^{-c \bullet (h-1)}$. □

The scheme requires only (s-1) • c additional keys instead of the t_{max}-1 of the key generation scheme of section 3.2.2.1.

The number of field operations per round is in the order of (s-1) • c - 1. To avoid unnecessarily expensive field computations, F = GF(2) should be chosen, therefore with $h \geq 2$, $\mathrm{Prob}_A \leq 1 / 2^c$.

Since each of the n participants should send a message within s slots, s should be in the order of n. Then the scheme requires O(n•c) operations. For F=GF(2) and therefore $c \approx -\log(\mathrm{Prob}_A)$ this is equal to $O(n \bullet -\log(\mathrm{Prob}_A))$.

3.2.2.3.2 Combination of probabilistic key generation and explicit tests

Assume that the probabilistic scheme of eq. (3.2) is used in combination with explicit tests.

By the same argumentation as above it follows that instead of t_{max} - 1 additional keys at most (s-1) • c are really necessary, i.e. it is possible to use the (s-1) • c keys $b^0, \ldots, b^{(s-1)\bullet c-1}$ cyclically:

Let $a^1, \ldots, a^{t_{max}}, e, b^0, \ldots, b^{(s-1)\bullet c-1}$ be randomly chosen keys. Then

$$\begin{aligned} K_{ij}{}^t &:= a^t + b^{t \bmod (s-1)\bullet c} \bullet K_{ij}{}^{t-1} + e \bullet I_i{}^{t-1} \\ K_{ji}{}^t &:= a^t + b^{t \bmod (s-1)\bullet c} \bullet K_{ji}{}^{t-1} + e \bullet I_j{}^{t-1} \end{aligned} \tag{3.4}$$

Lemma 3.6 The key generation scheme defined by equation (3.4) satisfies condition SS.Together with the additional rules for testing and disturbing it ensures the fail-stop property in a probabilistic sense:

$$\mathrm{Prob}_A \leq 1 - \left(1 - \frac{1}{|F|}\right)^{(s-1)\bullet c}$$

Proof. The first part is proved as in lemma 3.4. The worst case for the second part, i.e. the best case for an attacker, is that from all testing participants only the last two are honest. Then the attacker is successless iff the actual value of d (defined as for eq. (3.2)) is greater than (s-2) • c, and the test detects the disturbance. Hence

$$\mathrm{Prob}_A \leq 1 - \sum_{j=1}^{c-1} \Big(\mathrm{Prob}(d=(s-2)\bullet c+j) \bullet (1 - \frac{1}{|F|^j})\Big) - \mathrm{Prob}(d \geq (s-1)\bullet c) \bullet (1 - \frac{1}{|F|^c})$$

$$\leq 1 - \mathrm{Prob}(d \geq (s-1)\bullet c) \bullet (1 - \frac{1}{|F|^c})$$

$$= 1 - (1 - \frac{1}{|F|})^{(s-1)\bullet c-1} \bullet (1 - \frac{1}{|F|^c})$$

$$\leq 1 - (1 - \frac{1}{|F|})^{(s-1)\bullet c}$$

□

Again only large fields F (e.g. ld(|F|) ≈ 150) are suitable.

3.3 Final remarks on fail-stop broadcast

Superposed sending together with one of the discussed fail-stop key generation schemes (sect. 3.2.2) guarantees the desired unconditional sender and recipient untraceability.

If one tries to transform this nice theoretical result into a real communication network, a lot of practical problems must be solved, but none of them becomes really harder if fail-stop broadcast is used in addition to normal superposed sending.

For this consider the performance of superposed sending measured by

- the number of exchanged keys per transmitted message,
- its communication complexity,
- its computational complexity,
- and the reliability of the scheme.

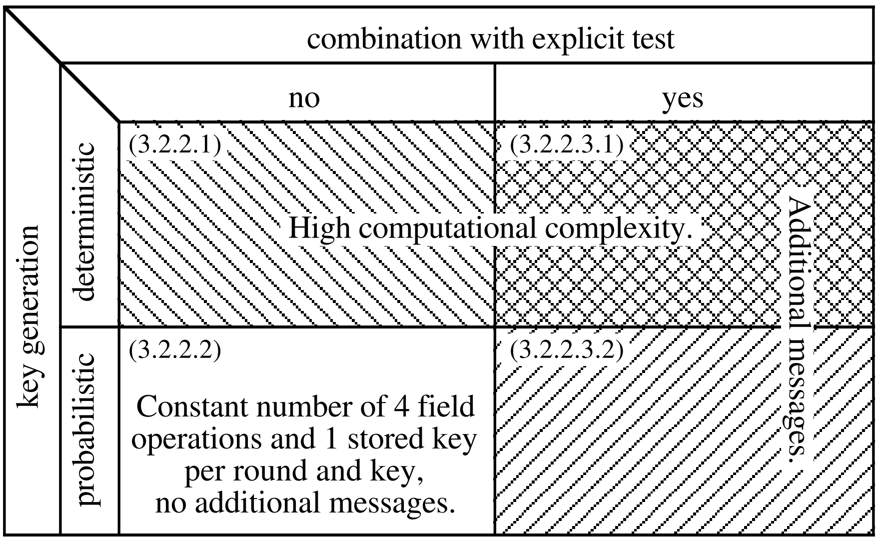


Figure 5 Comparison of fail-stop key generation schemes.

The **number of additional keys** is increased by at most a factor of two, which was shown to be the optimal value for deterministic key generation schemes without explicit tests. In theory this seems to be acceptable, and in practice one will mostly choose pseudorandomly generated keys anyway (and by this will loose the unconditional untraceability).

Communication complexity (Fig. 5)**.** None of the pure key generation schemes (sect. 3.2.2.1, 3.2.2.2) requires the sending of additional messages.

If combinations of key generation and explicit tests (sect. 3.2.2.3) are used, some honest participants may be forced to send meaningless test messages. The number of additional test messages depends on the participants' sending and testing rates. If real messages are end-to-end encrypted they appear to be randomly selected from F^c, i.e. they can be used instead of explicit test messages.

Computational complexity (Fig. 5)**.** The key generation requires some additional time and memory for each exchanged key. For that reason the schemes with deterministic key generation (sect. 3.2.2.1, 3.2.2.3.1) seem to be less practical, but if one uses one of the schemes with probabilistic key generation (sect. 3.2.2.2, 3.2.2.3.2), the computation requires only the storage of the last key and two field additions and multiplications per round and exchanged key.

All schemes except that of sect. 3.2.2.1 realize only *probabilistic* untraceability, i.e. there is a small probability that an attacker will successfully transmit different messages to different participants. But all four schemes don't rely on any unproved assumptions.

For probabilistic key generation (sect. 3.2.2.2, 3.2.2.3.2) only large fields F are suitable, but this is no hard restriction:

- Usually the cardinality $|F|^c$ of the set of all transmission units "message" will be relatively large. It doesn't matter whether one uses a small field and a large c or a large field and a small c.
- The reservation map technique described in [Pfi1_85 sect. 2.2.2.2] and (reservation by) superposed receiving (sect. 2.2) require a large cyclic group $(F,\oplus)$, anyway. It is important to notice that the group $(F,\oplus)$ used for superposed sending need not be the additive (or multiplicative) group of the finite field $(F,+,\bullet)$ used for key generation. E.g. one can use the field $F = GF(2^m)$ for key generation and, by interpreting the elements of $GF(2^m)$ as binary encoded integers, the additive group of integers modulo 2^m for superposed sending.

The **transmission delay** introduced by key generation could be decreased by parallelizing the key generation for different rounds, which can be done in two ways.

One can use $k > 1$ DC^+-nets, say $DC^+_0, \ldots, DC^+_{k-1}$, in a time division technique, i.e. in round t the DC^+-net $DC^+_{t \bmod k}$ is used. To preserve the untraceability each interaction between participants should be completely performed using only a single DC^+-net, i.e. each participant should answer a message only by that DC^+-net by which he has received the message.

The other possibility is to use only one DC^+-net, but to make the keys for round t not dependent on the directly preceding rounds $t-i$, $i = 1, 2, \ldots, t-1$, but on the rounds $t-i$, $i = k, k+1, \ldots, t-1$ for a $k > 1$. To preserve the untraceability each participant has to wait at least $k-1$ rounds before he answers to a received character.

Naturally the fail-stop property decreases the **reliability** of the network, since every inconsistent broadcast will immediately stop the network independent of whether it was caused by an attacker or a physical fault. But most of transient faults in a network can be tolerated by usual data link protocols [Tane_88 sect. 4], and if a permanent fault occurs (e.g. if a participant's station is damaged or all links between two participants are cut) superposed sending is disturbed and the network is stopped anyway. Therefore reliability is not essentially reduced by the discussed fail-stop schemes.

Hence, the pure probabilistic key generation scheme (sect. 3.2.2.2) with an appropriately large field F seems to be the most practical choice.

The problem of combining untraceability and **serviceability** in spite of active attacks is discussed in the following section 4.

4 Serviceability and untraceability

Up to now, we have a network which guarantees unconditional untraceability. We showed that the serviceability is not greatly reduced compared with the original DC-net. Nevertheless, without any further measures, the serviceability is not good, so there is reason to discuss whether it can be improved without giving up the unconditional untraceability. This has also been done in [Chau_88, section 2.5] very briefly, but the protocol proposed there contains a weakness, which we will remove, and needs a lot of refinements anyway. Also it relies on the assumption of reliable broadcast, like the original DC-net, but in contrast to superposed sending alone and the DC^+-net, and we will discuss how realistic this assumption is or can be made.

There are two possible causes for disturbance of superposed sending: faults and active attacks. Although we mainly discuss active attacks, some remarks on fault tolerance are in order.

1. There is no well defined difference between dishonest participants and participants whose stations are faulty. Hence for practical reasons the first simple disturbances of a station should always be viewed as a fault, whereas every notorious disturber should be viewed as an attacker and ultimately be eliminated from the DC^+-net. (One can try to make this difference a bit clearer by introducing mutually trusted devices: e.g. if the device looks like having been smashed with a hammer, probably the participant is an attacker, whereas if the device looks intact from outside, an unintentional fault would be assumed.)

2. As an active attacker can simulate or cause physical faults, measures against active attacks also help against faults. Nevertheless (because usually a faulty station or a faulty part of the network behaves stupidly) it should always be checked whether a network can be made more efficient if faults are first excluded by easier measures than those necessary against active attacks.

3. Fault tolerance mechanisms cannot be implemented without considering a potential active attacker: Otherwise they might offer the attacker a possibility for disturbing the DC^+-net or for tracing honest participants, e.g. by claiming that those were faulty.

For point 3. one can distinguish two classes of fault tolerance measures and corresponding faults:

The first class consists of measures which can be implemented in a way transparent for the DC^+-net and therefore don't affect the untraceability (i.e. these faults are "under" superposed sending if the DC^+-net is considered as a layered system). Examples for this class are

- error detection codes and appropriate data link protocols [Tane_88 sect. 4], which can be used to tolerate transient transmission faults, and
- fault masking techniques [AnLe_81] to tolerate most faults of the stations.

The second class consists of measures to tolerate those faults which could not be eliminated by the first class and therefore affect superposed sending directly: e.g. a station

- accepts an incorrectly transmitted value as input and therefore, according to the fail-stop key generation scheme, computes incorrect keys for all following rounds, or
- is not able to compute the outputs properly, or
- is not able to submit the outputs for the global superposition, e.g. because all links between it and the other stations are permanently cut.

All these faults have the same result: the DC$^+$-net is permanently disturbed. To recover the DC$^+$-net from such failures it is necessary to detect faults, to localize disturbing stations (which are nevertheless perhaps owned by honest participants), and to do some error recovery.

For some kinds of faults of the second class, all known techniques to tolerate them assume that the faulty station is eliminated from the DC$^+$-net until it is repaired or replaced. Therefore the set P of all *active* participants will be dynamically defined and it is important to ensure that

- the connectivity of $G \setminus (P \times A)$, i.e. the presumption of lemmata 2.1 and 3.1, is always satisfied, and
- the set of all honest participants $P \setminus A$ must always be large enough to satisfy the *intuitive* notion of anonymity.

If a faulty station is not eliminated from the DC$^+$-net it may behave arbitrarily. Therefore such faulty stations should be viewed as attacking, i.e. A consists of all attacking participants *and* of all participants whose station is faulty but not eliminated.

The untraceability of the DC$^+$-net should not be decreased. Unfortunately the fault tolerance techiques using elimination and reintegration of stations require the knowledge of which participants' stations are faulty and which others are repaired and will be reintegrated in the next round. This knowledge allows some new attacks similar to those described in section 3: If the attacker sends a message M and doesn't get the expected answer from his anonymous partner X he can argue that X was one of those participants whose stations have become faulty in one of the last rounds. Furthermore if M is the first message of a protocol the attacker can repeat M periodically. If at some time he gets an answer he can argue that X was one of the last reintegrated participants.

This attack cannot be detected since the attacker behaves like an honest participant, and it could only be prevented if this kind of faults were not tolerated, i.e. if the entire network is stopped until all faulty stations are repaired.

For the following (and in practice) it is assumed that this lack of untraceability is acceptable.

For concrete fault tolerance measures see [EcNi_89, Nied_87, MaPf_87, Pfi1_85 sect. 3.2, Pfit_89 sect. 5.4]. In the following, only active attackers are further considered.

4.1 Serviceability in spite of active attacks

Without any precaution each faulty or dishonest participant P_i can untraceably and enduringly disturb superposed sending by choosing $O_i \in_R F$ instead of according to eq. (2.1), the fail-stop key generation, and the multi-access protocol, etc.

In contrast to faults, active attackers cannot be localized in a special localization phase after a disturbance has been detected, because during such a phase a clever attacker would certainly stop disturbing. Also the localization must not take place for a disturbed slot, if someone else might have legitimately tried to send a message in it, because that could lead to the tracing of this message.

Therefore in [Chau_88 sect. 2.5] a protocol for laying "traps" and prosecuting attackers who get caught in them is suggested. The protocol (like the original DC-net) depends on the assumption of a reliable broadcast network.

In section 4.2 the protocol is repeated and it is shown that it is insecure (even under this reliable broadcast assumption): by a kamikaze attack the sender of a randomly selected message can always be identified. In section 4.3 the protocol is improved in a way which preserves the untraceability under the reliable broadcast assumption.

In section 4.4 it is shown that without the reliable broadcast assumption the protocol of section 4.3 can be used for successful attacks on the untraceability. Then the possibility of serviceability in spite of really unconditional untraceability is discussed and a (not very efficient) protocol developed which achieves this. In section 4.5 a protocol is described which guarantees serviceability and untraceability based on the reliable broadcast assumption and the assumption that there is an honest majority of participants.

4.2 The original protocol based on the assumption of reliable broadcast and how it can be misused

Throughout section 4.2 and 4.3, the existence of a reliable broadcast network is assumed, i.e. we assume that the attacker is not able to manipulate the consistency of broadcast. Thus the untraceability is not really unconditional. We also assume that this network allows each participant to determine the origin of each published message unambiguously.

The term *"P_i publishes message x"* means that P_i sends x to all other participants using the reliable broadcast network.

The problems discussed in section 3 are not posed any longer on this assumption, thus there is no need for fail-stop key generation. As mentioned in section 3.1, the assumption is not very realistic, see section 4.4.

As far as serviceability is concerned, the attacker is assumed to be computationally limited.

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

4.2.1 The original protocol by Chaum

The original protocol [Chau_88] assumes that the reservation map techique with GF(2) as group (cf. section 2.1) is used, and that each participant P_i reserves exactly one slot in each reservation phase. Before each reservation phase he decides whether to use this slot for sending a real message or for sending a **trap**, i.e. a meaningless message whose only purpose is to be disturbed by an attacker. If he decides to send a trap and he has chosen index k for the reservation, he announces this by publishing an encrypted version of the message *"I use the slot reserved by index k for sending a trap y"* . This message will be called "trap proof" in the following.

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

If the trap of P_i is disturbed, P_i publishes the trap announcement in clear together with the used encryption key (and for probabilistic encryption the coin tosses used for encryption).

Then 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 x_k which corresponded to index k. From these all publicly known outputs O_j^t can be checked, hence at least one attacking participant P_a can be detected: either he has correctly published his really used, but not allowed value $M_a^t \neq 0$, or he has modified at least one key K_{aj}^t for 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".

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

4.2.2 The basic attack

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

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

If it does not happen, he gives up for this time and waits for the next reservation frame. So far, the attack cannot be noticed, so the attacker can keep trying arbitrarily long. On an average, he will have to wait n/2 reservation frames for success, which is acceptable for him.

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

Then he publishes his trap proof for slot x_k and thus causes the deanonymization of the legitimate user of slot x_k. If this legitimate user has also published a trap proof for slot x_k 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 x_k, and the legitimate user will be identified by the prosecution protocol. On the one hand he has thus been traced as the sender of message y*, on the other he is unjustly punished as attacker.

4.2.3 Refinements of the original protocol and the remaining kamikaze form of the old attack

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. (So far, in the case that y* was no trap, there was no risk at all for the attacker.) Of course this means that round k of the reservation frame must be investigated.

Then the attacker will loose a key, but this is not sufficient for untraceability based on the reliable broadcast assumption: Even if only all secrets of the reservation round k, not of slot x_k, are published, the attacker can still identify the sender of message y*, because the attacker publishes his trap proof after y* has been sent, and the sender is the only person except for the attacker who claims to have reserved slot x_k.

The probability that this happens can be decreased if not all secrets are published at once, i.e. one tries to identify the attacker before the legitimate user has to publish his secrets. Thus the participant who published the trap proof should always be the first to publish all his secrets, and all published secrets should be discussed immediately. I.e. before any other secret is published, the partners with whom the keys are shared must publish whether they agree to the published values. The investigation should be terminated as soon as the first incorrectness or disagreement about a key has been detected.

If the attacker is only one participant, the other participants are now secure. But this cannot be assumed. For the case that there are several participants colluding as attacker, call the one who published the trap proof P_a. Since P_a has not reserved slot x_k, the prosecution protocol forces him to publish the wrong message character for the reservation round. The error made by this must be corrected by changing a secret key exchanged with another participant, and as the keys are immediately discussed, this must be an allied attacker, say P_b. If the real sender has to publish his secrets of the reservation round before P_b, he will be identified. Otherwise P_b will be assumed to an attacker and will at least loose a key from the key graph, or P_b can look for a third allied attacker P_c who has not yet published the key $K_{cb}{}^t$ and can change that key, etc. The attack is called "kamikaze" now, because in any case the attacker will at least loose a key each time he attacks.

4.2.4 An improved kamikaze attack for the refined protocol

As the attacker is now sure to loose a key each time he attacks, he might find it unsatisfactory not to be able to choose the message which he is going to trace. Therefore he could change his tactics and not send y in slot x_k, i.e. choose all his message characters as 0. Then if y* is a trap he runs no risk, and if it is an uninteresting message he can also save his key for future use. Even if secrets of the message slot are published after an inconsistency in the reservation round has been detected, this does not lead to complete unmasking of the attacker, as he can again change the keys, preferably the same he changed in the reservation round.

4.2.5 Influence of the reservation technique on the quality of the protocol

Similar protocols and attacks are conceivable for various reservation techniques, not only for the bit-map one. In this section an additional weakness of the protocol is discussed, which can be removed if another reservation technique is used, and it is checked that the original attack is not avoided if the other reservation techniques of section 2.2 are used.

4.2.5.1 A better kamikaze attack specially for the bit map reservation protocol

An additional weakness of the original protocol is that the reservation map techique with GF(2) as group does not allow 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).

Assume two colluding attackers P_a and P_b. First they publish two trap proofs for two indices k and k' of a reservation frame. (If the reservation frame to which a trap proof belongs cannot be seen from the encrypted version, they can even publish many more.)

Now both of them try to reserve the slots corresponding to the trap proofs, i.e. output a 1 in rounds k and k' of the reservation frame. Additionally, they choose the two indices k_a and k_b to reserve one slot for each of them. This behaviour cannot be distinguished from the correct one. In fact, if they have a key in common, they need not even agree on this behaviour in advance: they can just both lie about their keys in rounds k and k' later. In this last case, they cannot even be discovered if the reservation phase is investigated for some other reason.

Again, if nobody uses an index for which they published a trap proof, they just wait and repeat the attack.

If some time an innocent user is caught in one of the traps, say for index k and corresponding to slot x_k, both P_a and P_b use this slot too, i.e. they send messages y_a and y_b. Note that they can still see the message y* of the legitimate user. One of them then publishes the trap proof. Now both of them and the legitimate user show that they reserved this slot, so there is no inconsistency at all in round k of the reservation frame, nor in slot x_k. The only way to punish the attackers is by showing that they reserved more than one slot.

As both will claim that they didn't, it seems necessary that each participant publishes which slot he had reserved, and that the reservations for the remaining two slots are investigated to find an inconsistency.

Thus the attackers have not only traced one message, but all messages corresponding to one reservation frame.

This attack cannot be avoided by saying a priori that in such a situation two of the three participants who reserved the slot must be attackers, so one could remove all the keys these three share.

First, this would allow a simple kamikaze attack (although with little chance of success), where the attacker reserves two slots too much and hopes that three other reservations will collide; then those honest participants would loose keys.

Secondly, this new attack is not even kamikaze for an attacker who can break the commitment scheme (although success is still rare): The attacker could reserve three slots only in such situations where he has already found out that three others will collide. This situation is possible, as the attacker on untraceability is not computationally limited, at least if a commitment scheme is used in which the indistinguishability of trap and non-trap announcements is only computationally secure. (This is called "secure for the verifier" in [BrCC_88], and otherwise there is no reason against using such a scheme.) Then the resulting situation is indistinguishable from the one in which all participants who had no collision are honest and two of the three who suffered the collision are attackers; thus the real attackers cannot be punished.

It should be noted that if there were a practical application in which the possibility of successful kamikaze attacks could be tolerated and thus the original protocol with the refinements of 4.2.3 used (instead of a more secure, but slightly less efficient one), it is essential that reservation frames with less than n ones are not used. Otherwise collisions of three messages, one of them a trap, could occur by chance or an attacker could try to force them without being detected.

4.2.5.2 An attack on serviceability for the bit map reservation protocol

A completely different attack can be successful against serviceability: If there are several colluding attackers, in each reservation frame two of them can choose the same index, so that the reservation must be repeated. This could have happened by chance, so there is no good reason to punish them. If there are only a few attackers, so that the same pairs of participants collide very often, one would nevertheless have to punish them after a while, as then the probability that honest participants are punished can be made negligibly small. If there are many attackers (there can be up to n-2), the borderline between allowed collisions and collisions considered as attack would have to be carefully chosen.

4.2.5.3 Reservation map with large group

The same protocol and attacks as those described in 4.2.1 - 4.2.4 are possible when a larger group is used for reservation map (see section 2.2).

The attack of 4.2.5.1 is avoided if $m \geq n$ (n was the number of participants, m the modulus).

If $m \geq n$, reservation frames in which a few reservations collided can be used without allowing the attacks mentioned at the end of 4.2.5.1, as an honest user is always sure whether there was a collision with his own reservation or not. Therefore, also the attack of 4.2.5.2 would no longer be possible, i.e. the attackers (the computationally limited kind assumed for serviceability) could only force collisions among themselves, and those would not harm anybody else.

As the length r of the reservation frames can be chosen shorter than for $m = 2$, the time the attacker must wait before success is a bit shorter than for $m = 2$, but this does not matter much.

4.2.5.4 Reservation by superposed receiving

Still the protocols and attacks of 4.2.1, 4.2.2 are possible in principle. Nevertheless the situation is much better, as the probability that an honest user chooses the same reservation message RM as an attacker decreases exponentially in the length of the reservation messages, whereas the length of the reservation frames increases only linearly. Therefore the probability of success for the attacker can be made arbitrarily small.

The refinement of 4.2.3 is not possible, at least not without tracing more messages than the attacked one, as some of the reservation messages are never sent alone, but only computed from sums with others.

This can be changed if during the reservation those reservation messages which could be computed from others are nevertheless at the end repeated explicitly by their senders. The reservation phase stays deterministic, and its length increases by less than the factor two.

The attacks of 4.2.5.1 and 4.2.5.2 can be avoided just like in 4.2.5.3.

4.3 The improved protocol based on the assumption of reliable broadcast

The following protocol can be seen as an extension of that of [Chau_88 sect. 2.5]. Especially the idea of laying traps is kept up, but the trap announcements are linked with the traps in a better way. Also we refine several other parts of the protocol, which would be needed for the original protocol too (e.g. the "palaver phase"). Before the single parts are considered, we give an overview:

Assume a reservation protocol by which n' of all n participants have successfully reserved a slot. *Each* participant P_i who has reserved a slot x_i sends an encrypted **announcement** in slot x_i, and in slot $n'+x_i$ he sends a **trap** or a **non-trap**, i.e. a real message, according to his announcement.

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

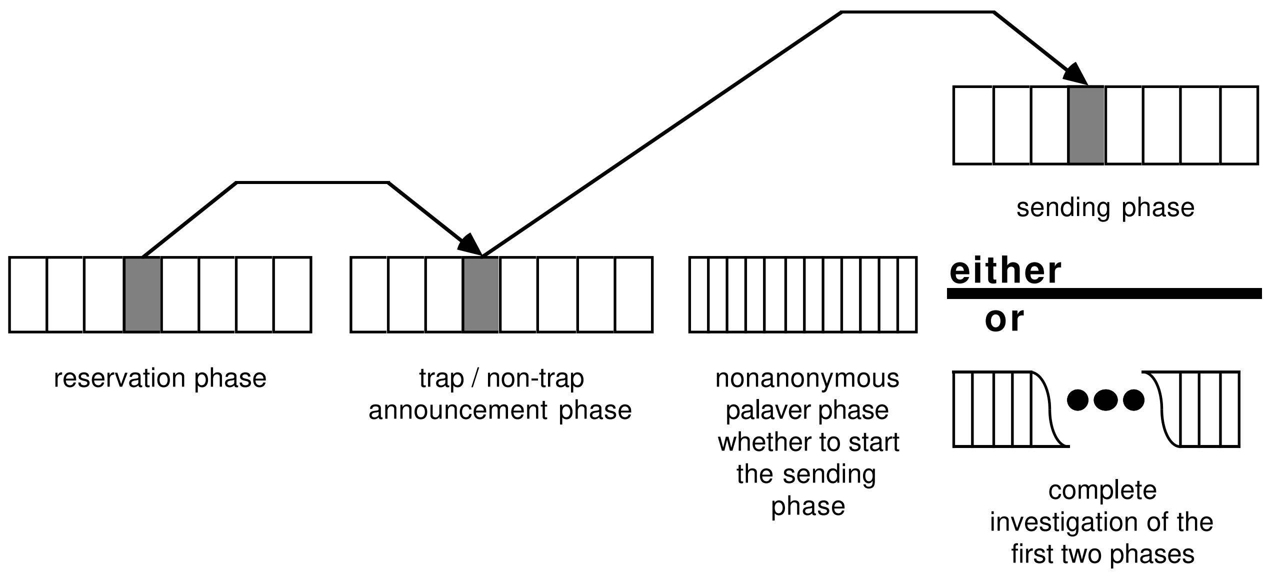


Figure 6 Phases of the improved protocol.

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. Once the sending phase has been entered, the reservation and announcement phases may no longer be investigated. So the reservation and announcement phases and the sending phase must be separated by a non-anonymous **palaver phase**, in which each participant who has detected a disturbance during the first two phases can prevent all others from entering the sending phase. In such a case it is possible to investigate the first two phases completely without decreasing untraceability. Otherwise, i.e. if the sending phase is entered, the first two phases will never be investigated. Disturbances during the sending phase are investigated only if the corresponding trap proof is given.

The structure of the protocol is depicted in fig. 6. It is described in full detail in the following sections.

As in section 4.2, the restrictions on the attacker are:

- for the basic version of our protocol
  - for untraceability: the reliable broadcast assumption
  - for serviceability: the reliable broadcast assumption and a computational restriction,
- in an alternative version
  - for untraceability and serviceability: the reliable broadcast assumption and $3 \cdot |A| < |P|$

(As discussed in section 4.3.1.2, if $3 \cdot |A| < |P|$ holds, a much weaker assumption about the underlying network implies the existence of reliable broadcast.

Another protocol which assumes the attacker to be restricted by the reliable broadcast assumption and $2 \cdot |A| + 1 < |P|$ for untraceability and serviceability is described in section 4.5.)

We will describe both versions in parallel and call the first one the version with **computationally secure serviceability**, the latter the version with **information-theoretically secure serviceability**.

One has to bear in mind that if an attacker succeeds in having an honest participant regarded as attacker and thus eliminated from the network, this is an attack against untraceability. (One can argue that this participant, not being able to send any longer, is perfectly untraceable. But at least the remaining honest participants are less untraceable than before.) Therefore in the version with computationally secure serviceability, for this kind of attack the attacker is only restricted by the reliable broadcast assumption. Of course, the same holds for the elimination of a key between two honest participants.

As in section 4.2, the term *"P_i publishes message x"* means that P_i sends x to all other participants using the reliable broadcast network. By assumption each participant can unambiguously determine the origin of each published message.

4.3.1 Outputs and output commitment

All outputs are published on the assumed reliable broadcast network. In all following situations the number of outputs needed from each participant is known a priori. So we can 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 the other participant's local outputs, i.e. each participant has to commit to his local output O_i^t for several rounds before knowing the other participants' outputs for any of these rounds.

Otherwise assume that the attacker is the first one who gets all local outputs, i.e. who can compute the correct global sum. Then he would be able to disturb the reservation phase by producing collisions, and during the sending phase he could disturb all obviously sensitive messages, e.g. all messages which are addressed to himself.

This **output commitment** can be done …

- For *computationally* secure serviceability by a computationally secure commitment scheme [BrCC_88]. Each participant distributes his commitment to all participants in a first phase. In a second phase the commitments are opened, by this distributing the outputs O_i^t.

  (To improve efficiency, P_i can use a single commitment for all O_i^t of one slot together. Also the x_i^t for reservation rounds could be shown only on demand, e.g. if a collision has occurred. But this is not possible for the sending phase, as then usually only the sender of a message would know about a disturbance and would thus identify himself by asking.)

  One can also use a one-way function f, as described in [Chau_88 sect. 2.5]. The value $f(O_i^t)$ is used as commitment for $O_i^t \in F$. Of course it must be ensured that F is large enough to avoid that someone compares all values f(x) with a commitment, or enough rounds must be grouped together for commitment.

  Naturally this is still no real commitment scheme as defined in [BrCC_88]: if the attacker can guess a special x, he can test whether a participant has committed to x. But in our case, for most f, this should not help him because of the superposed keys: even if the attacker suspects the message which might come out as global sum S^t and controls all participants except for two, P_i and P_j, who have a key in common, he is not able to test all |F| possible combinations of O_i^t and O_j^t.

  But there are efficient and insecure implementations of this scheme: Assume f is an exponentiation function of GF(p), p a large prime, and $F = \mathbb{Z}/_{(p-1)\mathbb{Z}}$. Inverting f is the well known discrete logarithm problem [Odly_85, CoOS_86, LoWi_88, Pera_86]. Then the product of all output commitments is equal to the commitment to the sum of all outputs, i.e. to the global sum. Thus if the attacker is the last one to publish his commitment, he can check in advance whether the global sum will be equal to an expected sensitive message.

This implementation may be secure if the attacker is not able to expect messages, e.g. because each character contains a large random part. Then the function may be useful in the case that the reliable broadcast assumption is implemented by Byzantine agreement: instead of immediately agreeing on everything, the participants agree only on the product of all commitments and the sum of all outputs. Only if product and sum are incompatible, or if the slot is investigated afterwards, they agree on all commitments and outputs afterwards.

- For *information-theoretically* secure serviceability the schemes of [BeGW_88, ChCD1_88] can be used for implementing **simultaneous broadcast** as defined in [CGMA_85].

  They guarantee untraceability only under the assumption $3 \cdot |A| < |P|$, too, as more attackers can give the wrong impression that an honest participant did not fulfil his commitment and must be eliminated. It's therefore clear that this version of our protocol cannot guarantee untraceability without the assumption $3 \cdot |A| < |P|$. (The same problem occurs in section 4.3.3).
- *physically*, as also mentioned in [Chau_88], e.g. if all outputs occur simultaneously on differnt frequencies of a network. One must bear in mind that this means not only that the participants simultaneously output one bit each, but that all participants output all bits of a slot at the same time.

Note that for computationally secure serviceability an attacker who can break the output commitment scheme does not endanger the untraceability, only the serviceability. Thus there is no special need to choose a commitment scheme which is unconditionally secure for the prover or the verifier (cf. [BrCC_88]): If the attacker (as prover) can open commitments in two ways, he is not bound to his own commitments. If (as verifier) he can decrypt the commitments of other participants, he can choose his own commitments depending on theirs, if he has to publish his last. In both cases this is just the original problem of serviceability.

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 network. Because of the reliable broadcast assumption, this cannot happen to him if he is honest. Also the other participants can decide about elimination locally, and all honest participants will get the same result.

4.3.2 Reservation phase

Recommended reservation behaviour: To be able to investigate the first two phases without loss of untraceability (before entering the sending phase), it is necessary that during the first two phases no participant uses any sensitive information. Therefore the reservation behaviour in the first two phases must be independent of the participants' real sending wishes, i.e. some participants will reserve message slots without using them, and some other participants 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. Obviously, instead of only one message, each P_i can reserve a fixed number of traps and of non-traps.

Allowed reservation behaviour: It must be clear in which situations a reservation phase is investigated, and when a participant is considered as attacker.

The easiest way to fix the *allowed behaviour* is to force each participant to reserve a fixed number of slots each time. In the following we assume that this number is 1.

With respect to untraceability, one could also have replaced the fixed numbers by fixed reservation *rates*. This is not advisable with respect to serviceability, as it would allow attackers to send more messages on an average than honest participants, because in each single reservation phase this could not be proved.

It must also be regulated what happens to *reservation collisions*.

All reservation techniques described in section 2.2 are probabilistic, i.e. for each of them it can happen to two (honest) participants to reserve the same slot: For reservation map techniques they may choose the same index, for reservation by superposed receiving they may choose $RM_i = RM_j$.

So a collision cannot be considered as 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* (sect. 4.3.1). For reservation map techniques, one should protect a complete reservation frame together (protecting each round alone gives the attacker a slightly better chance to disturb the last rounds and is usually less efficient). For reservation by superposed receiving the slots of a reservation frame can of course not be protected together (because they depend on each other). One could be tempted to protect only the first slot, because this first one determines all following reservation slots. But of course this holds only if all participants are honest. Attackers who share keys only have to decide upon the sum of their reservation messages in the first slot, because they can make up for any internal difference later by claiming they had other keys.

Output commitment does not prevent attackers from causing collisions among themselves, like in 4.2.2.5. To avoid the problem of setting a limit for the number of allowed collisions, which always leaves a chance that an honest participant is considered as attacker and eliminated, one should not consider these as attack either. As they cannot be prevented either, this means that they must be accepted, i.e. a reservation phase is not investigated just because of collisions. (Thus these attackers only harm themselves.)

Reservation frames with an *impossible result* (i.e. one which could not have occurred among honest participants) are investigated. Globally, the allowed results are

- for the reservation map technique with $m \geq n$: the sum of the messages of all rounds is n (in $\mathbb{Z}$)
- for the bit-map reservation protocol: the number of reservations is $n-2 \cdot k$ for some $k \in \mathbb{N}_0$
- for superposed receiving: there is a collision of n messages initially, and the numbers and sizes of the messages in the following rounds fit.

Additionally, each participant can make tests depending on his knowledge about his own reservation:

- For the reservation map technique with $m \geq n$ he knows that there cannot be a zero where his own reservation should be.

- For reservation by superposed receiving he knows that he must receive his own reservation message.

The local test for the reservation map technique is not necessary, as an attacker who cannot break the output commitment scheme and tries to cause this situation will with great probability not hit a reservation and then be caught by the global test.

Also, during superposed receiving the sizes of the messages need not be checked; it suffices to check that the protocol can be carried through, ends after n rounds (or earlier, if there are collisions), and that each participant checks that he received his own reservation message.

During the palaver phase, each honest participant who saw that one of the tests does not hold will vote for investigating the first two phases.

All tests are secure for the tester, i.e. if they indicate a fault, the investigation (cf. 4.3.5) will find a protocol deviation. Thus if the investigation doesn't find any incorrectness, the tester can be assumed to be an attacker.

Problems with the bit-map reservation protocol: If one keeps the idea that for unconditional untraceability there may not even be a very little chance that an honest participant is regarded as attacker and eliminated, the bit-map reservation protocol should not be used:

As discussed above, a reservation phase is not investigated just because of collisions. Especially if the bit-map reservation protocol is used, this implies that undetected collisions of an uneven number of messages can occur. These will usually result in a disturbance during the announcement phase. It can be seen from section 4.3.3 that in this protocol (in contrast to the original one, cf. the end of 4.2.5.1) this does not harm untraceability.

Nevertheless it can be used to attack serviceability. There are two conceivable ways to implement the announcement phase: either a disturbance of an announcement after a reservation frame with collisions is investigated or not. In the first case, at least three participants attacking together can always cause a collision of three of their reservations and thus invalidate the announcement phase, thus this implementation cannot be chosen. In the second case they can cause a collision of two of their reservations and later claim that these two collided with one of an honest participant (this cannot be distinguished); then they can disturb the announcement of this participant. For the protocol version with computationally secure serviceability, this cannot be accepted at all, because for $|A| > 2/3 \cdot |P|$ the attackers can disturb all announcements (always two attackers are needed to disturb one other participant). For the version with information-theoretically secure serviceability this could be accepted in principle, as $3 \cdot |A| < |P|$ guarantees that the attackers cannot disturb all announcements in this way, but it seems not trivial to invent a palaver scheme which decides whether too many announcements have been disturbed without giving up the anonymity of those participants whose announcements were disturbed, and anyway the possibility of disturbing some of the announcements is undesirable.

Synchronization of the phases: For each reservation technique described in section 2.2 each participant can locally determine the length of each reservation phase (for the reservation map techniques 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 have a fixed length, the exact durations of the following announcement and sending phase are determined. (This is necessary, or at least some other way by which each participant can locally and information-theoretically unambiguously determine where each announcement and each message ends.)

4.3.3 Announcement phase

Announcements are implemented by using a **bit commitment scheme** [BrCC_88]:

- Each participant P_i can commit to an encrypted announcement, i.e. to one of the values "trap" and "non-trap", and can later "open" any encrypted announcement he has committed to, i.e. he can give a trap proof which convinces each other participant P_j of the correct value "trap" or "non-trap". Hence it is not possible to open an encrypted announcement both as "trap" and as "non-trap".
- No other participant is able to decrypt an encrypted announcement, i.e. there is no (efficient) way to distinguish between encrypted "trap" announcements and encrypted "non-trap" announcements.

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.

Information-theoretically secure serviceability: Information-theoretically secure announcements (called *multi-party blobs* in [BrCC_88]) can be realized by applying the techniques of [BeGW_88, ChCD1_88].

Since an unexpectedly numerous attacker (i.e. $3 \cdot |A| \geq |P|$) can open non-trap announcements as trap announcements, this version of our protocol cannot guarantee unconditional untraceability. (The same problem occured in section 4.3.1.)

Computationally secure serviceability: Computationally secure bit commitment schemes can be divided into two classes depending on which part of the definition is unconditionally realized.

The first class of commitment schemes perfectly conceals the difference between trap and non-trap announcements (and is called secure for the *prover* in [BrCC_88]). From this it follows that each such announcement can be opened in two ways, to announce a trap and to announce a non-trap. Hence the correctness of a given proof can only be computationally secure, i.e. a very powerful attacker would be able to publish a trap proof for an arbitrary sensitive message, and the unconditional untraceability would be lost. (If the real sender publishes his non-trap proof, too, he is still identified, but the attack is detected and therefore seems to be acceptable in practice: one can switch from the broken bit commitment scheme to another, still unbroken one, if there is any.)

The second class is the opposite of the first one (and is called secure for the *verifier* in [BrCC_88]): each announcement can be proved only in one way, i.e. a given trap proof is unconditionally correct and it is guaranteed that no sensitive message will be traced by the prosecution protocol. But the indistinguishability of trap and non-trap announcements is only computationally secure (and, at the moment, in the best case polynomially equivalent to some well known hard problem, e.g. computing discrete logarithms [BrCC_88 sect. 6.2.2]). In practice it may be reassuring that announcements have to withstand only for a relatively short time (until the corresponding slot of the sending phase), but a very powerful attacker will be able to disturb non-traps only and thus will not be identified by the prosecution protocol .

A commitment scheme of the second type can be implemented by a probabilistic encryption scheme for encrypting the set {trap, non-trap}[GoM1_86]:

For appropriate sets X, Y let π: {trap, non-trap} $\times X \rightarrow Y$ be a public encryption function generated by P_i for committing to the value $v \in$ {trap, non-trap}. The first argument is the plain text, the second provides the necessary "coin-tosses". The pair $(\pi, \pi(v,x))$ is used as commitment, it is opened by showing the pair (v, x). Since decryption is unique, each commitment can only be opened in one direction [BrCC_88 sect. 6.2.3].

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

Each honest participant P_i whose announcement was disturbed will vote for investigating 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 (see 4.3.2 for the problems if the bit-map reservation protocol would be used). Thus the test is secure for the tester.

4.3.4 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 detects something wrong and votes for "investigation", the first two phases will be completely investigated and the reservations become invalid. Otherwise the sending phase is entered.

For voting (like for other outputs), there can be a fixed order among the participants. This ensures that each participant gets a chance to vote, and that the origin of each vote is clear, so that somebody who disturbs serviceability by wrongly declaring reservation phases as invalid can be punished.

The votes should only consist of one bit each. If one wants that the participant who detected a disturbance tells more precisely what it was, this should be postponed to the beginning of the investigation, as otherwise it would unnecessarily reduce efficiency in the faultless case.

4.3.5 Investigation of reservation and announcement phase

If any participants vote for 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.

(For many cases the investigation can be shortened if the participant who detected a disturbance describes it more precisely, but here we will not discuss this further. As far as only active attacks are concerned, it is also of no great importance, as the guarantee for serviceability implies that investigations are necessary only a finite number of times, until all attackers or all their keys are eliminated. If physical faults are considered, which occur more frequently, this can be different.)

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

Firstly, for all rounds t and participants P_i, P_j the rules of superposed sending are checked, i.e. it is tested whether equation (2.1) is satisfied for the values published by P_i, and whether $K_{ij}^t = K_{ji}^t$ for each $\{P_i, P_j\} \in G$, where K_{ij}^t is published by P_i, K_{ji}^t by P_j. Deviations from (2.1) prove "$P_i \in A$", deviations from $K_{ij}^t = K_{ji}^t$ prove "$P_i \in A$ or $P_j \in A$".

In the first case P_i is eliminated, i.e. each participant discards all keys exchanged with P_i, and all messages sent by P_i will be ignored in the future. In the second case the edge $\{P_i, P_j\}$ is eliminated from G. As in the original protocol, in case only one of P_i and P_j is an attacker, this should not harm the other one, as a key shared with an attacker is not secure anyway. (This is slightly idealized, because in practice there might be different groups of attackers who don't work together).

The diagnosis "$P_i \in A$ or $P_j \in A$" can be refined by applying signatures [Chau_88]: for this P_i and P_j certify each key $K_{ij}{}^t$ a priori by privately exchanging corresponding signatures. If round t is investigated, together with $K_{ij}{}^t$ they publish their partner's signature, and each participant who cannot publish a valid signature is assumed to be an attacker. Obviously an unlimited attacker can forge signatures, thus it may happen that both participants can publish "valid" signatures; then only the key K_{ij} can be removed from G.

Secondly, the observance of the reservation protocol is tested.

Correct reservation is described by the reservation protocol itself and rules for the allowed reservation behaviour stated in 4.3.2. As all the tests of 4.3.2 only check for results which are impossible if the reservation protocol was correctly executed, one is sure to find an incorrectness now, if one of those tests failed and no inconsistency was found when checking the rules of superposed sending.

Thirdly, the announcement phase is tested. This is quite easy since a correctly executed reservation protocol determines who was allowed to use which slot. (It is not necessary to open any announcement, i.e. its value trap or non-trap.)

Each found incorrectness of a participant P_i is used as a proof for "$P_i \in A$" and P_i is eliminated from the net.

If no inconsistency is found, the participant P_i who has initiated the prosecution protocol is viewed as attacking and eliminated from the network. This is correct, since all tests which allow to vote for "investigate" during the palaver phase are secure for the tester (sect. 4.3.2, 4.3.3).

Since faults and simple active attacks are undistinguishable (cf. the beginning of section 4) in practice it would be useful to give each attacker at least a second chance, i.e. one would first try to resynchronize or repair his station etc. Only key pairs which are inconsistent so often that unintentional faults seem unlikely are eliminated from the key graph, and only notoriously attacking participants are eliminated from the net.

4.3.6 The sending phase and investigation of traps

If no participant voted for "investigation" during the palaver phase, the sending phase is entered and investigation of the first two phases becomes taboo (i.e. the secrets; of course the announcements can be opened).

Sending: All slots of the sending phase are protected by output commitment (sect. 4.3.1). The commitment must always been done for a complete slot in advance, not for single bits, as 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 not able 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 $\mu/(1+\mu)$.

Showing trap proofs: Slots are only checked if the corresponding trap proof is given, i.e. published an arbitrary (but for practical reasons limited) time after the trap was sent. Each participant must get a chance to publish trap proofs, so some of the bandwidth of the reliable broadcast channel must be reserved for him and this purpose. E.g. there can be trap-palaver phases like those of 4.3.4 every now and then; the trap proof itself can be given at the beginning of the following investigation.

Investigation: After a trap proof has been published, first the rules of superposed sending are checked like in 4.3.5.

Secondly, 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: The computationally unlimited attacker against untraceability could otherwise decrypt the trap-proofs of other participants and publish them first. Then the real trapper would be regarded as attacker and eliminated from the network.)

This can be done most easily if a trapper is forced to send nothing (i.e. zeros) in his own trap. Then any participant who sent anything else is an attacker. This does not reduce serviceability because of the output commitment.

For the reservation map technique, but not for reservation by superposed receiving, it is alternatively possible to investigate the corresponding reservation round. For reservation by superposed receiving one can either use the variant mentioned in section 4.2.5.4, or one can investigate the corresponding announcement slot instead, because if that was not investigated after the announcement phase, it is clear that only the trapper sent in it. Only in the rare case that the announcement consists of zeros only, this does not help (and this again could be prevented by suitable coding). (The announcement could also be investigated for the reservation map technique, but there it gives no additional information about the attacker.)

Again a participant who unjustly initiated the prosecution protocol (by publishing a trap proof for an undisturbed slot) is viewed as an attacker and eliminated from the net. As every disturbance can be proved, this cannot be used to eliminate honest participants.

4.4 Removing the reliable broadcast assumption

Without the reliable broadcast assumption the trap and prosecution protocol described in sect. 4.3 cannot be applied without giving away *unconditional* untraceability:

An unlimited attacker could accuse an honest participant P_i by forging his output O_i^t and use the prosecution protocol to oust P_i or at least to eliminate a key P_i shares with another perhaps honest participant from the key graph. 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 would be lost. (As mentioned in 4.3, this is clear for the case that two honest participants loose their common key. If an honest participant is eliminated altogether, one can argue that his untraceability is not lost, because he cannot send anything to be traced. But the untraceability of the remaining honest participants is clearly reduced.)

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

In section 4.4.1 the reliable broadcast assumption is directly justified physically or by using Byzantine Agreement protocols. Naturally the untraceability remains "conditional".

Section 4.4.2 suggests an informal definition for guaranteeing serviceability while preserving untraceability, and section 4.4.3 describes how a scheme can be implemented which guarantees "nearly" unconditional untraceability (i.e. untraceability if the attacker cannot prevent communication betweentwo honest participants) and computationally secure serviceability.

4.4.1 Implementing reliable broadcast

4.4.1.1 Physically implemented reliable broadcast

Here we just mention some possibilities: we neither claim completeness nor really assess their security.

The easiest implementation of reliable broadcast is the acoustic implementation. It might be quite sufficient for the original dining cryptographers (see [Chau_88]) sitting at their dinner table, if the music in the restaurant is not too loud. For more general applications reliable acoustic broadcast can also be used in the form of a court. But, as already mentioned, in the protocols under consideration this court would not only be needed in the case of faults, but for every single output of each participant. Thus this implementation seems rather unsatisfactory.

The more common physical implementations use electrical broadcast media. As we cannot expect that each participant controls a satellite, ground radio could be the preferred implementation, but busses might do, if each participant can constantly physically check that the bus has not been partitioned.

The protocols also assumed that the origin of each message could be unambiguously determined and that each participant got a chance to send. E.g. this could be achieved by a fixed division of the bandwidth of the network (by frequency or time) and the use of an x-out-of-y-code, if one can assume that a disturbance can only transform a 0 into a 1, not a 1 into a 0 (or any other code for this purpose, like Berger codes [Prad_86]. But this doesn't work if the attacker can send about half a one, so that a 0 of the original sender will arrive as 0 at some participants and as 1 at some others.

4.4.1.2 Reliable broadcast by Byzantine Agreement

Reliable broadcast is usually realized by Byzantine Agreement protocols (cf. section 3.1).

The obvious method for publishing a message is to distribute it to all participants, who will then agree on the message by using an appropriate Byzantine Agreement protocol. The attacker is restricted either by $3 \bullet |A| < |P|$ or by some computational limitations. Which of the two possible restrictions is assumed must usually be decided a priori (we show a new protocol in which this is not necessary in section 5.1).

For asurvey of lower bounds for reliable broadcast and known solutions see [Reis_87], an efficient randomized protocol can be found in [FeMi_88].

The basic net is always assumed to be synchronous (mainly because otherwise it could not be decided whether someone disturbed the protocol by not sending at all, or whether the message has only not yet arrived).

It is also assumed that the attacker is unable to prevent the communication between honest participants. The part that he cannot cut off the communication completely must be realized physically. The part that messages which arrive are correct can be realized cryptographically, in case one is willing to accept that with very small probability the assumed attacker can nevertheless be successful. Then one can use perfect authentication codes [GiMS_74, Sim3_88] (or nearly perfect codes [WeCa_81]).

For the case that the attacker is computationally restricted, i.e. cannot forge signatures, one might be tempted to try to improve efficiency by a central implementation. Then each participant would send his signed output only to one centre instead of to all participants, the centre would perform the addition and distribute the results. In case of disturbances one would hope to be able to resolve disputes between the centre and the participants by the signed values. The problem with this implementation is that it cannot be decided whether a participant sent nothing at all or the centre suppressed it.

If the fail-stop key generation is used for superposed sending, each attempt of an unexpectedly numerous or powerful attacker to prevent the reliable broadcast stops superposed sending and initiates the prosecution protocol. Thus the scheme guarantees *unconditional* untraceability as long as the prosecution protocol is not initiated. Additionally the scheme guarantees unconditional untraceability as long as $G \setminus A \times P$ remains connected, but this condition cannot be verified by the honest participants, i.e. the untraceability becomes "conditional" as soon as the first key is eliminated from the key graph G.

4.4.1.3 Using centres as representatives

A weaker method for realizing reliable broadcast is to use a number of centres $C_1, \ldots, C_m$ as **representatives**.

It is assumed that each honest participant can reliably communicate with each centre. To reliably broadcast a message a participant P_i sends his message to all centres C_j, which together guarantee that the message is distributed to all participants:

If the centres are able to reliably broadcast messages to all participants (e.g. by a satellite for each centre), it suffices to assume that

- either each participant signs his messages, the attacker is not able to forge signatures, and at least one centre is honest, or
- that there is an honest majority of centres.

Otherwise the centres agree on the message received by P_i (either by physical broadcast or Byzantine Agreement) and each centre distributes the result of the agreement to the participants. From all received messages, each honest participant selects that one which was distributed by the majority of all centres. Then it suffices to assume that the centres are able to reach agreement and that there is an honest majority of all centres (which is necessary for information-theoretically secure Byzantine Agreement anyway). This is similar to the techniques for Byzantine Agreement with less the maximum number of tolerable attackers [DoSt_83].

If the untraceability is based on trustworthyness of the centres, it is sufficient that each participant exchanges secret keys only with them [Chau_88 sect. 2.3].

One can consider to use the centres in other parts of the protocol, too.

4.4.2 What "guaranteeing serviceability while preserving untraceability" means

Assume an implementation of the DC^+-net which guarantees untraceability in spite of an attacker who is limited by a predicate $\mathbf{A_{untr}}$, e.g. $A_{untr} = \emptyset$ (i.e. unconditional untraceability), or A_{untr} = "A is not able to break signatures".

Then a protocol is said to **guarantee serviceability** in spite of an attacker who is limited by an assumption $\mathbf{A_{serv}}$ equivalent to or weaker than A_{untr} if the following two conditions hold:

S1 *Serviceability:* If the attacker A satisfies A_{serv} then after a finite number of disruptions A will loose at least one key from the key graph, and no pair of honest participants will loose a common key.

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

The protocol of section 4.3 guarantees serviceability for A_{untr} = "reliable broadcast assumption" and $A_{serv} = A_{untr} \wedge$ "the attacker is computationally restricted" or $A_{serv} = A_{untr} \wedge 3 \bullet |A| < |P|$.

The implementations described in section 4.4.1 assume one of the conditions for reaching Byzantine Agreement for both untraceability and serviceability.

Condition S2 describes a *fail-stop* property similar to that of section 3.2. One may therefore consider condition S2 to be unnecessarily weak, since e.g. the protocol of section 4.3 satisfies the stronger and more natural condition

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

The reason why we have chosen S2 nevertheless is that satisfying S1 and S2 together seems easier in some cases, and that if A_{untr} holds, but not A_{serv}, seviceability is not guaranteed anyway, thus another reason for the honest participants to stop participating does not harm, and that if both A_{serv} and A_{untr} hold, $S2^*$ is implied by S1 and S2 anyway.

4.4.3 Fail-stop Byzantine Agreement

In the following, an idea is presented which transforms each Byzantine Agreement protocol which works with any kind of signatures (as far as we know, all computationally secure Byzantine Agreement protocol fulfil this) for tolerating up to n-2 attackers into a protocol which

- guarantees agreement provided the attacker is not more powerful than assumed, and

- allows each honest participant whose signature was broken by an unexpectedly powerful attacker to *prove* this to each other participant in an unconditional way.

This is achieved by a new signature scheme, which allows participants (with very high probability) to prove if their signatures are broken.

Hence soon after the first signature of an honest participant 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 the communication between honest participants. This seems to be the weakest possible assumption and is therefore called **nearly unconditional**.

Using fail-stop Byzantine Agreement in the protocol of 4.3 obviously satisfies condition S2 (section 4.4.2). Since our solution guarantees the fail-stop property in an unconditional way, we have combined nearly unconditional untraceability (with very high probability) and computational serviceability.

Unfortunately our solution is not very efficient (yet ?).

4.4.3.1 A signature scheme whose forgery can be proved

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

- The probability of unprovable forgery can never be zero:

  An honest user must be able to produce at least one signature for each message from the message space. Thus if the (computationally unlimited) attacker who can break the scheme finds just this signature, it cannot give the honest user additional information. So he cannot prove that the signature was broken (otherwise he could also deny the signature which he himself would have produced, which cannot be tolerated).

- There must be more than one possible signature for each message, i.e. more than one value must be acceptable to the other participants (although perhaps the honest participant will only be able to produce a single one). This must hold unconditionally.

  This has just the same reason as the previous point: Only a signature that the computationally limited user could not have produced himself may allow him to prove that it is not his. Thus such signatures must exist, and the computationally unlimited attacker must be unable to distinguish them from the ones the honest participant can produce.

- It is not possible to prevent a dishonest, computationally unlimited participant from denying his signatures, i.e. "proving" that they were broken, even though this is not true:

  Having the same possibilities as if he were an attacker against himself, he can produce the signatures which he could not produce if he were computationally limited, and then he can prove that they are broken.

  But in all our applications this is entirely right, as this situation means that there is an attacker who can break the signature scheme, and so one cannot further rely on these signatures anyway.

Thus one is looking for a signature scheme in which most forged signatures allow the supposed signer to compute something that under the assumption of the signature scheme he could not have computed before.

The idea of our scheme is that the signatures consist of square roots, and forged signatures allow to factor the modulus. Thus (in contrast to the Rabin scheme [Rabi_79]) the factorization of the modulus cannot be known to the signer in advance. The scheme to which this lead us is very similar to one-time-signatures. Therefore we start by shortly describing these.

4.4.3.1.1 One-time signatures

One-time signatures are an easy method for signing a limited number of bits using a one-way function f, attributed to Lamport in [DiHe_76, Merk_88]:

If at most k bits are to be signed, the signer chooses 2•k elements $r_{0,1}, r_{1,1}, r_{0,2}, r_{1,2}, \ldots, r_{0,k}, r_{1,k}$ and makes the sequence

$$f(r_{0,1}), f(r_{1,1}), f(r_{0,2}), f(r_{1,2}), \ldots, f(r_{0,k}), f(r_{1,k})$$

public. To sign the bit sequence $(b_1, \ldots, b_k)$, he publishes

$$r_{b_1,1}, r_{b_2,2}, \ldots, r_{b_k,k}$$

A slight improvement of the scheme is described in [Merk_88] (attributed to Winternitz there), but nevertheless the scheme seems to be rather inefficient.

An efficient variation of the theme is described in [Merk_88], and in [BeMi_88] the idea is used to construct cryptographically strong signature schemes using a strong trapdoor function generator. Both improvements cannot be used in the following.

4.4.3.1.2 Cryptographically strong one-time signatures whose forgery is provable

Idea: As mentioned, the idea of our scheme is that signatures are square roots of known values modulo composite numbers, because then knowing several signatures of one message, i.e. several square roots of the same values, gives a good probability that one can factor the modulus. The factorization of a modulus will thus serve as a proof that something is wrong.

Of course, for this proof to be convincing, the modulus cannot have been chosen by the signer. In fact, the only easy situation in which this proof really proves to someone that someone else can factor (i.e. leaves no possibility that someone has only published a previously known factorization) is that the verifier of the proof has chosen the modulus himself.

For simplicity, we first describe the resulting scheme for just one signer and one verifier, and then generalize it to the situation in which all participants sign and verify each other's signatures.

Protocol for two parties: Assume two parties A, B, and call them Alice and Bob for convenience (and according to tradition). Our goal is to construct a one-time signature scheme which allows Alice to sign a k-bit-message for Bob. If someone else forges a signature of Alice, she is able to prove to Bob that the signature scheme is broken. Additionally, even if Bob himself forges the signature, she can prove to a third party Vera that either the signature scheme is broken or Bob is among the attackers.

Let σ be the **security parameter for factoring**, i.e. that value for which it is assumed that factoring a product of two primes, each of length $\sigma/2$, is not feasible for the attacker.

A second security parameter d, the **security parameter for the probability of unprovable forgery**, is chosen.

Basic protocol

[1] Bob chooses two large primes p, q each of length $\sigma/2$ and sends the product $m := p \cdot q$ to Alice together with a proof that it is composite (e.g. a witness according to the probabilistic primality test of Rabin-Miller [Rabi_80]).

[2] Alice checks that m is in fact a composite number.
She randomly chooses $2 \cdot d \cdot k$ different elements

$$r_{0,1,1}, \ldots, r_{0,1,d}, r_{1,1,1}, \ldots, r_{1,1,d},$$
$$\ldots\ldots$$
$$r_{0,k,1}, \ldots, r_{0,k,d}, r_{1,k,1}, \ldots, r_{1,k,d}$$

of $\mathbb{Z}/_{m\mathbb{Z}}$ all coprime to m, and makes their squares

$$s_{b,x,y} := (r_{b,x,y})^2$$

modulo m public.

[3] To sign the bit sequence $(b_1, \ldots, b_k)$ Alice publishes the corresponding roots

$$r_{b_1,1,1}, \ldots, r_{b_1,1,d},$$
$$\ldots\ldots$$
$$r_{b_k,k,1}, \ldots, r_{b_k,k,d}$$

[4] Bob verifies the received roots.

Explanation: Of course, squaring modulo m corresponds to the one-way-function of the original one-time-signatures. Note that line x of the r-matrix serves to sign the x-th bit of the message; the left half of the r-matrix serves to sign zeros, the right half to sign ones; and the fact that there are d square roots for each bit instead of one increases the probability that a forgery enables Alice to factor m.

Now assume a forth party, Felix, the forger.

To forge a signature which Bob will accept in step [4], Felix has to compute at least d square roots modulo m (because he must give a new signature for at least one bit). Computing square roots and factoring are equivalent problems [Rabi_79, Woll_87], thus our basic protocol can be viewed as a cryptographically strong authentication code based on the intractability assumption of factoring. (The proof that m is composite should not help in this, as every other participant could have found it for himself. We put it into step [1] instead of [2] only to ensure that Alice cannot disturb the protocol by claiming that she is very sorry not to find any witness. One can also trade expected efficiency for certainty that a proof is found by using the algorithm of [AdHu_87] in step [2], or by letting Bob give Alice a zero-knowledge proof between the two steps [GoM1_86].)

Since Bob knows the factorization of m he can tell it to Felix, enabling him to compute square roots of all $s_{b,i,j}$ efficiently. But since all roots chosen by Alice are coprime to m, each quadratic residue published by Alice has two significantly different square roots (i.e. two roots r, r' with $r \neq \pm r'$, see e.g. [Kran_86, Theorem 4.5]), and even Bob cannot suspect which one Alice has chosen. (This is why Alice had to check that m is not prime; if it has more than two factors her probability is even better.)

Thus in any case if Felix has forged Alice's signature, i.e. he has changed at least d roots of the really signed message, with probability not less than $(1 - 2^{-d})$ he has chosen a significantly different root for at least one of Alice's squares.

Call this root r' and call Alice's root r. Then Alice can factor m using r, r', since $(r-r') \bullet (r+r') = 0 \pmod{m}$. If Bob is honest, this proves to him that someone (Felix or Alice) can factor. Also it proves to Vera that either someone can factor or Bob has wrongfully disclosed his factorization.

Thus we have proved the following lemma.

Lemma 4.1 Assume the basic protocol for two parties Alice and Bob described above.

i. The problem of forging an authenticated message which Bob will accept in step [4] is equivalent to the integer factoring problem.

ii. Each forged signature enables Alice to factor Bob's modulus m with probability not less than

$$1 - \frac{1}{2^d} .$$

Proof. see above. $\square$

The scheme can naturally be extended to n parties $P_1, \ldots, P_n$, each of whom wants to sign a k-bit-message.

Basic protocol for n parties

[1] Each participant P_i chooses two primes p_i, q_i of length $\sigma/2$ and makes their product $m_i := p_i \bullet q_i$ public together with a proof that m_i is composite.

[2] Each participant P_j checks that each m_i is in fact a composite number.

Then for each P_i participant P_j chooses $2 \bullet d \bullet k$ values coprime to m_i like in the two-party-protocol, i.e.

$$r^{(i,j)}_{0,1,1}, \ldots, r^{(i,j)}_{0,1,d}, r^{(i,j)}_{1,1,1}, \ldots, r^{(i,j)}_{1,1,d},$$
$$\ldots\ldots$$
$$r^{(i,j)}_{0,k,1}, \ldots, r^{(i,j)}_{0,k,d}, r^{(i,j)}_{1,k,1}, \ldots, r^{(i,j)}_{1,k,d}$$

of $\mathbb{Z}/m_i\mathbb{Z}$ all coprime to m_i, and makes their squares

$$s^{(i,j)}_{b,x,y} := (r^{(i,j)}_{b,x,y})^2$$

modulo m_i public.

[3] To sign a specific k bit message, P_j makes the corresponding $(n-1) \bullet d \bullet k$ square roots public, i.e. for all $i \neq j$:

$$r^{(i,j)}_{b_1,1,1}, \ldots, r^{(i,j)}_{b_1,1,d},$$
$$\ldots\ldots$$
$$r^{(i,j)}_{b_k,k,1}, \ldots, r^{(i,j)}_{b_k,k,d}$$

(this corresponds to one signature for everybody else in the two-party-protocol).

[4] Every P_i with $i \neq j$ verifies the received $(n-1) \bullet d \bullet k$ roots.

Each forged signature enables P_j to factor the modulus of a special other participant with probability $(1 - 2^{-d})$. Thus with probability not less than $(1 - 2^{-d})^{n-1}$ she is able to publish a factor p_i of each m_i. Thus we can supplement the basic protocol:

[5] Each honest participant who receives a signature of P_j sends it back to P_j.

[6] If P_j receives a forged signature, she factors all m_i if she can. Then she publishes one non trivial factor of each m_i as proof that someone has forged a signature.

[7] Each honest participant who receives the factorization of *all* moduli accepts this as a **"proof of forgery"** (more precisely: as proof that someone can break the signature scheme, not that someone has forged anything). All signatures become invalid.

As for the basic protocol for two parties we get the following lemma:

Lemma 4.2 Assume the basic protocol for n parties described above.

i. The problem of forging a signature which an honest participant will accept in step [5] is equivalent to the integer factoring problem.

ii. Each forged signature enables the supposed signer P_j to factor all moduli m_i, i.e. to form a "proof of forgery", with probability not less than

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

Conversely, forging a "proof of forgery" is equivalent to factor all n public moduli.

iii. The signature of a k bit message comprises $k \cdot (n-1) \cdot d \cdot \sigma$ bits.

Proof. see above. □

It is not obvious at this place why we emphasized the fact that participant P_i waits to see all moduli factored before he accepts it as "proof of forgery". Of course, if he is honest, he is already sure that the signature scheme is broken if he sees his own m_i factored. The necessity that a proof of forgery is something that convinces all participants will become clear when the scheme is used within Byzantine Agreement, because there this fact serves to ensure agreement upon whether the scheme was broken or not.

The participants can also try to choose one m together instead of all the m_i, as this would reduce the length of the signatures and the number of exchanged squares by a factor of n. If one requires that at the end of the choosing protocol each participant locally checks that a correct m has been chosen (like in step [2] of the above protocol), this does not reduce the probability that the supposed signer of a forged signature can form a "proof of forgery". If the choosing protocol is suitably chosen, it should hold that a computationally restricted attacker cannot influence the protocol such that he knows the factors of m a priori, thus if a correct m is found at all, forging signatures or a "proof of forgery" should still be difficult. However, finding m can only be guaranteed for $2 \cdot |A| < |P|$ [GMW_87].

4.4.3.1.3 Using iterated squares to improve efficiency

The main problem of the signature scheme of section 4.4.3.1.2 (protocol for n parties) is that for signing each single bit one has to prepare $2 \cdot (n-1) \cdot d$ squares and distribute them. In the following we try to improve this ratio by using iterated squares in two ways.

In [Merk_88] a method of Winternitz is mentioned to sign one out of k+1 different values using any one-way-signature scheme and only two prepublished values: for this the signer chooses two values r, s, and publishes $f^k(r)$ and $f^k(s)$. To sign the value j, $0 \le j \le k$, the signer makes public $f^j(r)$ and $f^{k-j}(s)$.

As in this method the amount of computation (i.e. the number of applications of f) grows exponentially with the length of the message signed in one piece, one can reduce the number of values to be distributed a priori only by a logarithmic factor.

To apply this to the scheme of 4.4.3.1.2, for each m define the function

$$f_m(x) := x^2 \pmod{m}.$$

To be able to use this improvement we have to ensure that nobody can decide which of the two significantly different roots of $f_m{}^k(r)$ is $f_m{}^{k-1}(r)$, thus we have to choose m specially:

Lemma For moduli $m = p \cdot q$ with $p = q = 1 \pmod 4$, $a \in (\mathbb{Z}/m\mathbb{Z})^*$, and $k \ge 2$, all square roots of $f_m{}^k(a)$ (i.e. all possible values for $f_m{}^{k-1}(a)$) are quadratic residues.

The modulus m is appropriately chosen iff -1 has a square root modulo m.

Proof. Let QR(z) be the set of all quadratic residues modulo the integer z, and let $a \in (\mathbb{Z}/m\mathbb{Z})^*$. Let $b := f_m{}^{k-2}(a)$, thus $f_m{}^k(a) = b^4$.

Then $f_m{}^k(a)$ has four distinct roots, namely the two roots b^2, $-b^2$ and two other roots r, -r.

By Euler's criterion $-1 \in QR(p)$ and $-1 \in QR(q)$, thus $-1 \in QR(m)$. Thus b^2 and $-b^2$ are both quadratic residues modulo p, q, m. Since b^2 and r are both roots of $f_m{}^k(a)$ modulo p, $r = b^2$ or $r = -b^2 \pmod p$, thus $r \in QR(p)$. The same holds for root -r and for modulus q.

Therefore all four roots of $f_m{}^k(a)$ are quadratic residues. $\square$

1st method: Thus to use Winternitz' method in the basic protocol, Bob has to choose an appropriate modulus and to publish a root of (-1). Alice chooses arbitrary $r_1,\ldots,r_d$, $s_1,\ldots,s_d$ coprime to m and publishes the $f_m{}^k(r_x)$, $f_m{}^k(s_x)$.

A precondition for this is that these moduli may not be much easier to factor than other products of two large primes. Also it is presupposed that the root of -1 doesn't make factoring easier, but this could be changed by using a zero-knowledge-proof instead.

The generalization to the protocol for n parties is obvious.

2nd method: If the security assumption about this kind of modulus holds, for special applications of the signature scheme (including our intended application in protocols for Byzantine agreement), a similar, more valuable efficiency improvement is possible: Assume that time is partitioned into phases, that the signer has to sign only a certain number of bits for each phase and that these signatures becomes meaningless as soon as the current phase is finished.

Then it is possible to prepublish just $2 \cdot d$ values for signing bit position x in all k phases together, namely the $f_m{}^k(r_{b,x,y})$ for some chosen $r_{b,x,y}$. In phase k' (k' = 1,…,k), for signing the value b for bit x, the signer publishes the values $f_m{}^{k-k'}(r_{b,x,y})$.

To avoid that the amount of computation to check signatures grows with the phase number, at the end of each phase the signer can also publish the unused signatures.

Of course, the two methods can be combined.

In our applications, a phase will be one Byzantine agreement.

4.4.3.2 The protocol of Dolev and Strong with an efficiency improvement for our signature scheme

In [DoSt_83] Dolev and Strong describe (and prove) a computationally secure protocol for Byzantine Agreement, which is able to tolerate up to n-2 attackers and forces each honest participant to send at most two different messages.

Each message of the protocol consists of the value the sender has broadcast and of at most 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 protocol uses iterated signatures (like $s_{i_k}(s_{i_{k-1}}(\ldots s_{i_2}(s_{i_1}(v))\ldots))$). This would be very unpleasant for our inefficient signature scheme, as iterated signatures would grow exponentially with the number of signers. Luckily, the iterated signatures can be implemented by *sets* of signatures instead. These grow only linearly with the number of signers.

Because of this change and because the protocol description given in [DoSt_83] is rather short anyway, we present the proof of Dolev and Strong for our scheme:

For convenience let P_1 be the sender. All messages of the protocol are sets of signatures, i.e. they are of the type $\{ s_{i_k}(v), s_{i_{k-1}}(v), \ldots, s_{i_2}(v), s_{i_1}(v) \}$. A message (set) is called **consistent** if all signatures contain the same value v and a signature of the sender P_1 is among them.

A participant P_j **relays** a consistent message by adding his own signature to it and forwarding it to all participants whose signatures do not yet appear in it.

Byzantine Agreement protocol for up to n-2 attackers (Dolev, Strong)

[1] The sender P_1 signs his value $v \in F$ and sends $\{s_1(v)\}$ to each other participant.

For k = 1,…,n-2:

[k+1] Let $V_{i,k}$ be the set of all consistent messages P_i has received during the k-th phase and which are signed by exactly k different participants, not including himself. Assume that $V_{i,k}$ is totally ordered.

If P_i has not relayed any messages during the previous phases and $V_{i,k} \neq \emptyset$, he relays the first two messages in $V_{i,k}$ with distinct values, or the first message, if all messages in $V_{i,k}$ contain the same value.

If P_i has relayed only one message during the previous phases containing a value v' during all previous phases and $V_{i,k} \neq \emptyset$, he relays the first message in $V_{i,k}$ which contains a value $v'' \neq v'$, if there is one.

Once a participant has relayed two distinct values, he stops processing messages for the protocol and at the end he will decide "sender fault".

[End] If P_i has received exactly one value v during all phases together (in a consistent message with enough signatures), he decides to accept this value. Otherwise he decides "sender fault" and chooses the default value for v.

Lemma 4.3 [DoSt_83 Theorem 3] Provided signatures are unforgeable and the attacker is not able to prevent the communication between honest participants, the Dolev/Strong Protocol guarantees Byzantine Agreement within n-1 phases, i.e.

i. after phase [n-1] (i.e. in step [End]) each honest participant chooses the same value for v, and

ii. if P_1 is honest, each honest participant chooses the value sent by P_1.

Each participant relays at most two messages. The length of messages sent in phase k is O(k•sig) bit, where sig is the length of a single signature.

Proof. First assume the sender P_1 to be honest. Then each honest participant receives the same value $\{s_1(v)\}$ during phase [1], and this is the only value v which is signed by P_1. Thus after phase [n-1] all honest participants decide to accept v.

Now assume that P_1 is dishonest. If no honest participant receives any consistent message during the n-1 phases, they all agree on "sender fault" in [End].

Now assume that an honest participant P_i receives the consistent set

$$\{ s_{i_k}(v), s_{i_{k-1}}(v), \ldots, s_{i_2}(v), s_{i_1}(v) \}$$

in phase [k]. First it will be shown that either each honest participant will decide "sender fault" independent of v, or that after phase [n-1] each honest participant will have received v.

If k = n-1, then all other honest participant have already received $s_1(v)$, because otherwise they would not have signed v.

Assume k < n-1. If P_i relays the set, each other honest participant will receive it in phase [k+1], because it is still consistent and has one signature more. Thus assume that P_i doesn't relay the set, which may have two reasons:

1st P_i has relayed the same value in a previous round.

2nd P_i has already relayed two different messages.

In the first case each honest participant has already received v, and in the second each honest participant decides "sender fault" independently of v.

Thus if an honest participant P_i has received two values and thus decided "sender fault", all others have received them too, or they have decided "sender fault" anyway.

Consequently, if the honest P_i has received exactly one value v, no other honest participant can have received two. So, as they have received v, too, all decide to accept v.

Finally, if the honest P_i has received no value v, then so has everybody else according to the previous two paragraphs. Thus they agree on "sender fault". □

Each participant has to send at most two messages, and particularly he has to sign at most two values or $2 \bullet \lceil ld(|F|) \rceil$ bit. If the signature scheme of section 4.4.3.1.2 is used, each participant has to choose $(n-1) \bullet 4 \bullet d \bullet \lceil ld(|F|) \rceil$ squares.

Thus the scheme requires $n \bullet (n-1) \bullet 4 \bullet d \bullet \lceil ld(|F|) \rceil \bullet \sigma$ bits to be published in advance.

4.4.3.3 A protocol for fail-stop Byzantine Agreement

Assume that the Dolev/Strong protocol (sect. 4.4.3.2) together with the signature scheme of section 4.4.3.1.2 is used. (The same supplements and lemmata hold for each other Byzantine Agreement protocol based on signatures which tolerates up to n-2 attackers.)

Then we get **Byzantine Agreement in which breaking can be proved** by appending the following two phases to the Dolev/Strong protocol:

1st supplement to the Byzantine Agreement protocol

[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_i has received a forged signature $s_i(v')$, he tries to use the forgery for factoring the public moduli m_j of all other participants. If he is successful, he sends the factorizations of all m_j (including of his own m_i) to all other participants.

[End] If P_i has not found or received the factorizations of all public moduli m_j, he decides as in the original Dolev/Strong protocol. Otherwise he decides "signatures broken".

Lemma 4.4 **Byzantine Agreement in which breaking can be proved.** Assume the supplemented Dolev/Strong protocol implemented with the signature scheme of section 4.4.3.1.2, and assume $|A| < n-1$.

i. If the attacker is not able to forge signatures (i.e. to factor the public moduli), the protocol finds the correct agreement after phase [n+1].

ii. If an honest participant decides to accept the value v after phase [n+1] (due to the original decision rule given in section 4.4.3.2), then with probability at least

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

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 in which breaking can be proved**.

Proof. i. Since the attacker cannot forge signatures, the protocol works as the original Dolev/Strong protocol, thus lemma 4.3 guarantees the agreement.

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

Contradicting lemma 4.4 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 P_j. Thus in phase [n] P_j receives the forgery and in phase [n+1] he sends the factorization to all other participants with probability $(1\text{-}2^{-d})^{n-1}$. Hence in phase [n+1] P_i receives the factorization and decides "signatures broken", which contradicts our assumption. Thus P_i has accepted the correct value v.

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

iii. If an honest participant decides "signatures broken", he knows the factorizations of all public moduli m_j and can show them 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 those participants who know that something is wrong can prove it later.

Now assume that the almost secure 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 acts as sender.

Then we add the following rule to the protocol:

2nd supplement to the Byzantine Agreement protocol

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

[k] If in phase [k-1] P_i receives the factorizations of all public moduli m_j for the first time, he immediately decides "signatures broken" for *all* following (and the current) broadcasts and sends the factorizations 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 [k], each other honest participant does so in phase [k+1].

Hence we have realized **fail-stop Byzantine Agreement**.

Lemma 4.5 **Fail-stop Byzantine Agreement.** Assume that the Dolev/Strong protocol with the first and second supplement and implemented with the signature scheme of section 4.4.3.1.2 is iterated ad infinitum, and assume $|A| < n\text{-}1$.

i. If the attacker is not able to forge signatures (i.e. to factor the public moduli), the protocol realizes reliable broadcast.

ii. If an attacker, who can forge signatures, disturbs the t-th broadcast during the Dolev/Strong protocol, then with probability at least

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

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

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

Proof. i. Follows from lemma 4.4(i).

ii. This case means that the attacker sends the forged signature of an honest participant P_i to another honest participant. Thus with the stated probability, each of them receives the factorizations from P_i in phase [n+1] of the t-th broadcast, decides "signatures broken" and stops sending.

iii. In this case, the attacker either sends the factorizations to an honest participant directly in a phase k for the first time, or a forged signature of an honest P_i to P_i in phase k=n. If $k < n+1$, each honest participant decides "signatures broken" at the latest in phase [n+1], and the (t+1)st broadcast doesn't even begin. If $k = n+1$, the decisions according to the original Dolev/Strong protocol are correct, and each honest participant receives the factorization at the latest in phase [1] of the (t+1)st broadcast. □

4.4.3.4 Using fail-stop Byzantine Agreement for untraceability and serviceability

Together with the protocol of section 4.3 (i.e. used in all places where the reliable broadcast network was needed, fail-stop broadcast guarantees computationally secure serviceability while preserving nearly unconditional untraceability (i.e. the attacker on untraceability is only assumed to be unable to prevent the communication between honest participants) in the sense of S2:

The only way how two honest participants can decide differently is that one has accepted a value v while the other has decided "signatures broken" (lemma 4.4 ii). But then all honest participants will stop after the first phase of the next broadcast (lemma 4.5 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 of a participant who has not received any incorrect value. This cannot give him additional information.

Unfortunately with our inefficient signature scheme fail-stop BA requires at least additional $O(n^2 \cdot d \cdot \sigma \cdot \lceil ld(|F|) \rceil)$ bits which must be published in advance (this is an upper bound if the second method of section 4.4.3.1.3 is secure, as then after each agreement about a value O_i^t, i.e. after a constant number of signatures, the published values can be reused).

As this scheme is much less efficient than the schemes of section 3.2, in practice one could try to use the fail-stop broadcast of 3.2, and only if that is disturbed continually, one would change to fail-stop BA (by hand, as then one would have to distribute the squares first etc.).

Fail-stop Byzantine Agreement could also be used together with superposed sending to realize probabilistic fail-stop broadcast as defined in section 3, but the untraceability is only nearly unconditional, and naturally the schemes of section 3.2 are much more efficient for this purpose.

4.5 A protocol without commitment

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

In this section we describe a protocol without commitments, which only assumes reliable broadcast and $2 \cdot |A| + 1 < |P|$.

Also there will be an exponentially small probability that an honest participant is considered as attacker.

Serviceability will not be guaranteed in the sense of S1 in section 4.4.2, i.e. a clever attacker can undetectedly disturb in certain situations. But the network guarantees the honest participants a certain bandwidth, with which they can send undisturbed.

The protocol has a similar phase structure as that of section 4.3, although the non-traps are distinguished from traps in another way.

Before describing the phases, we describe a more generally usable technique, which we will use to allow two participants, who are anonymous towards each other, to communicate secretly and reliably, although we have no computational restrictions.

4.5.1 Key-less cryptography and authentication

In 1983 Alpern and Schneider proposed a scheme, **key-less cryptography**, which allows two participants P_i, P_j to exchange a secret key over a public network without using any secret information, provided that this network guarantees sender untraceability [AlSc_83].

Using superposed sending together with a reservation technique, this idea can be implemented in a very efficient way [Bura_88, Pfit_89]: Assume that participant P_i wants to send something to participant P_j in round t. For this, one of them must have reserved round t and they must have agreed on it.

Then in round t, P_i sends his message character M_i^t, and P_j sends a randomly chosen character $M_j^t \in_R F$, which is called a **masking character**. From the global sum

$$S^t = M_i^t \oplus M_j^t$$

only P_j can recover character M_i^t sent by P_i. Each other participant gets no information about M_i^t, i.e. it is perfectly concealed.

Key-less authentication can be realized by a simple authentication code [Bura_88]: Let $F = GF(p^k)$ be a finite field with $p \neq 2$. Together with M_i^t, P_i sends the check character

$$M_i^{t+1} := (M_i^t)^2$$

Both characters are perfectly concealed by two masking characters $(M_j^t, M_j^{t+1}) \in_R F^2$. The only chance of a manipulating attacker P_a is to choose and send a character $M_a^t \neq 0$ and an appropriate M_a^{t+1}. The manipulation is undetected if

$$(M_i^t \oplus M_a^t)^2 = (M_i^t)^2 \oplus M_a^{t+1}$$

$$\Leftrightarrow \qquad M_i^t = \frac{M_a^{t+1} - (M_a^t)^2}{2\, M_a^t}$$

Assume that the message character M_i^t is uniformly distributed in F. Since the equation unambiguously determines the secret character M_i^t, the attacker's probability of successfully forging an authenticated character $M_i^t \oplus M_a^t$ is less than or equal $\frac{1}{|F|}$. This is optimal, since it is equal to the redundancy of the code and since the secret key used for authentication is chosen from the set F^2 [GiMS_74].

4.5.2 Outputs

Like in section 4.3.1, all outputs are published on the assumed reliable broadcast network, and it has also been decided a priori when each participant has to publish which output.

As we have no scheme for output commitment under our assumptions, to prevent the attacker from disturbing selected messages, the participants must take turns in who is allowed to make his outputs last. To see that this helps, note that if the last participant is honest and shares at least one key with another honest participant, the attacker has no information about the resulting global sum at the time when he must choose his own output.

Of course, like for the commitments, usually the outputs of several rounds, e.g. of one slot, must be grouped together for this purpose. Then the attacker cannot guess from the global sums of the first rounds whether the resulting message will be worth disturbing.

The last participant could change at the beginning of each reservation frame. Then one is sure that the serviceability guaranteed by the following measures against random disturbances holds in at least |P| - |A| out of |P| executions of the protocol. This ratio is at least 2/n, but usually much better.

In the following, for serviceability we will only consider protocol executions in which an honest participant outputs last. (This fact is of course not known to the other participants, but if they repeat disturbed reservations or messages, it is guaranteed that they can only be disturbed a finite number of times.)

4.5.3 Reservation phase

Except for the output commitment, the reservation phase is identical to that of section 4.3.2. Wherever in section 4.3.2 several rounds were grouped together for output commitment, the same rounds are grouped together for output here. Also the tests whether reservation was undisturbed are the same and the reasons why they are secure for the tester (as nothing depended on the assumptions that we dropped in section 4.5).

4.5.4 Announcement phase

For convenience, call the participant who reserved the corresponding slot X, because he is anonymous.

This phase consisted mainly of commitments in the protocol of section 4.3. The known commitment schemes cannot be used on the assumptions of section 4.5. Luckily, a commitment is not really necessary for trap/non-trap announcements (because if X changes his mind about whether it is a trap or not, this does not harm anybody else, as the others are not allowed to disturb it anyway).

So in this protocol, this phase is only used to provide X with means which later allow him, and him alone, to declare whether it was a trap or a non-trap. (Thus the name of this phase, chosen in analogy to section 4.3, is a bit misleading this time, as in reality nothing is announced yet, not even necessarily decided).

For this purpose, each participant P_i sends a password W_i to X, which later allows X to prove his authorization towards P_i. To ensure that only X receives the password, and that he receives the correct one, key-less cryptography and key-less authentication (cf. 4.5.1) must be used, as X and P_i have no key in common for secret key cryptography or perfect authentication codes; otherwise P_i would know who X is.

As all P_i must send a password (even X himself, as otherwise he would be identified), P_i need not be untraceable during this action. Thus there can be a fixed order among them for this purpose. (It has nothing to do with the order for outputs.)

Thus:

- A slot of the announcement phase consists of n mini-slots
- In the i-th mini-slot, P_i sends a randomly chosen password W_i followed by the check character, and X superposes two randomly chosen masking characters.

If X does not receive a valid codeword in one of the mini-slots, he will vote for investigation in the following palaver phase. As this can only happen if either the mini-slot was disturbed or P_i did not send a valid codeword, and X has reserved the corresponding slot, this test is secure for the tester.

4.5.5 Palaver phase

After the announcement phase, there is the same non-anonymous palaver phase as in section 4.3.4.

4.5.6 Investigation of reservation and announcement phase

As in section 4.3, no sensitive information was used so far. So if any participant votes for investigation during the palaver phase, the reservation and announcement phase are completely investigated.

Exactly as in section 4.3.5, first everybody publishes all his secrets, then the rules of superposed sending and then the observance of the reservation protocol are checked, and deviations are punished.

Then the announcement phase is tested. A correctly executed reservation protocol determines who was X for which slot and was thus allowed to superpose masking characters. Also it is clear a priori which P_i has to send a password in which mini-slot. Thus each incorrectness can be attributed to a specific participant.

Again, if no inconsistency is found, the participant P_i who has initiated the prosecution protocol is viewed as attacking and eliminated from the network. This is correct, since all tests which allow to vote for "investigate" during the palaver phase are secure for the tester (sect. 4.5.3, 4.5.4).

4.5.7 The sending phase and investigation of traps

If no participant voted for "investigation" during the palaver phase, the sending phase is entered.

Sending: The only difference to simple superposed sending is that each participant must output all outputs of one slot together (to replace the output commitment, see section 4.5.2).

Showing trap proofs: In this protocol, publishing the passwords which correspond to a given slot is considered as trap proof for that slot and the wish that it should be investigated.

Each participant must get a chance to publish trap proofs, so we assume that directly after the slot there are n mini-slots on the DC-net, which X can use to publish the n passwords W_i in their original order. If X sent a non-trap or nobody was caught in his trap, he sends zeros in all these mini-slots.(

Disturbances during the mini-slots for trap proofs: If X sends the passwords (not if he sends zeros!) and is disturbed in one of the mini-slots, he can ask for an investigation of these mini-slots, because he only sent a trap and is therefore willing to be identified. For this purpose, there is a non-anonymous palaver phase after each group of mini-slots (i.e. one corresponding to each slot), in which each participant can vote for investigation of these mini-slots.

If the message was a non-trap, this does not lead to tracing of the sender, since he did not use the mini-slots.

Investigation of mini-slots: During the investigation of the mini-slots, as usual first all secrets are published and the rules of superposed sending are checked. Then the rules for sending in the mini-slots are checked:

If nobody, except possibly the participant who voted for the investigation, sent in the mini-slots, he is viewed as attacker. This cannot happen to the honest X, as he would only ask for investigation if there was a disturbance.

If more than one participant sent in a mini-slot, one must decide between the legal sender X and the attacker. This cannot be done by investigating the corresponding reservation slot, as both senders might be attackers and the message a non-trap. Instead, the decision can be made according to the passwords: If someone legally sent in a mini-slot, he must have published all the passwords. Thus an honest participant P_i can consider each other participant P_j who did not send W_i in the i-th minislot, but sent anything in any mini-slot, as attacker. So he can give up the key they shared (for all following rounds). Therefore with high probability (because he might guess the passwords) this attacker will lose all keys he shared with honest participants, if he had any left.

Punishment: If a participant decides to give up a shared key, he publishes this using the reliable broadcast network (there must be bandwidth reserved for this purpose). Thus the new key graph is known to everybody.

Now we use the assumption $2 \cdot |A| + 1 < |P|$: As the key graph was fully connected at the beginning, and as no pair of honest participants ever lose their key, any participant who has lost at least half his keys must be an attacker. Therefore the following addition to the usual punishments is safe:

Any participant who has lost at least half his keys is excluded from the network altogether. (✻)

Thus especially an attacker who disturbed the publication of passwords is excluded from the network at once with high probability.

Decision about investigation of the trap itself: Slots should only be checked if the corresponding trap proof is given. In our case, each participant P_i can only decide whether he received his own password W_i. To come to a global view, after each successful mini-slot phase each participant non-anonymously publishes whether he thinks that there will be an investigation.

An attacking participant P_i can claim that he did not receive his password and is thus not willing to investigate. Then the investigation cannot be performed, as not all honest participants can distinguish this situation from the other situations: E.g. if X is an attacker, different honest P_i can come to different decisions. Also, if X is honest, attackers can claim that they received their passwords when in fact they did not.

What can be done is that X, if he still shares keys with any of the P_i who are unwilling to investigate, renounces these keys. As X was still anonymous when the P_i had to declare whether they would take part in an investigation, and each attacking participant who is still in the network has at least one key in common with an honest participant (because of (✻)), he will lose a key with probability greater than 1/n.

(This is the reason why the passwords had to be published on the DC-network, not on the reliable broadcast network.)

Investigation of the trap: If all participants agree that the trap should be investigated, this is again done by first publishing the secrets and checking the rules of superposed sending. Then one has to distinguish trapper and attackers. This is easy now: Either one can rely upon the order of the participants (in the place of output commitment) and force trappers to send zeros only. Or the reservation phase can be investigated. In contrast to 4.3.6, it is also clear that the participant who published the trap proof, i.e. the passwords, is the trapper.

If an attacker has guessed all passwords correctly (which can happen only with a probability exponentially decreasing with the length of the passwords) and published them as trap-proof, the legal sender is either identified by the prosecution protocol, or, if he refuses to investigate, the attacker will recognize him by this.

5 Adaptive Byzantine Agreement

The signature scheme of section 4.4.3.1.2 can also be used for a Byzantine Agreement protocol in which one need not decide between the two restrictions on the attacker (computationally restricted or $3 \cdot |A| < |P|$) a priori. Instead it will always work if at least one of them is fulfilled.

The first idea is, of course, that one starts with a Byzantine Agreement protocol BA1 which works with any kind of signatures, implemented with the signature scheme in which breaking can be proved, and one has another Byzantine Agreement protocol BA2 which works for $3 \cdot |A| < |P|$ as alternative.

Now, in contrast to section 4.4.3.3, one needs a definite moment when everybody knows whether the signatures are assumed broken and BA2 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 we can then (also in contrast to section 4.4.3.3, where we had to consider the unlimitied attacker against untraceability) assume that the attacker is restricted by $3 \cdot |A| < |P|$.

This motivates the following (simple, not most efficient) protocol:

Basic Adaptive Byzantine Agreement protocol

[α] Perform a Byzantine Agreement protocol BA1 with signatures, implemented with the one-time signature scheme of section 4.4.3.1.2, and the 1st supplement.

[β] Perform a Byzantine Agreement protocol BA2 without signatures.

For i = 1,…,n:

[γ_j] Participant P_i broadcasts a decision message D_i using protocol BA2, namely either the factorization of all moduli, if he had them in [End] of step [α], or the value "no". (In each round of the protocol, a participant checks whether a value claimed to be the factorizations is correct. If not, he interprets it as "no".)

[End] If an honest participant P_j has decided that all D_i are "no", he decides for the value he received in step [α]. Otherwise he decides for the value received in [β].

Lemma 5.1 **Adaptive Byzantine Agreement.** If signatures are unforgeable or $3 \bullet |A| < |P|$, the adaptive Byzantine Agreement protocol guarantees Byzantine Agreement within a predefined number of phasess. More precisely, with probability at least

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

i. in step [End] each honest participant chooses the same value for v, and
ii. if the sender P_1 is honest, each honest participant chooses the value sent by P_1.

Proof. We distinguish three cases:

1st case: Signatures are unforgeable, i.e. nobody can factor all moduli.

Then step [α] finds the correct agreement (Lemma 4.4). It remains to be shown that each honest participant decides for this value in step [End]. This is obvious, because nobody is able to send factorizations in any of the steps [γ_j], so all honest participants will agree on "no" in all of these steps.

2nd case: The attacker disturbs step [α], i.e. he forges at least one signature of an honest participant or sends all factorizations to at least one honest participant.

Then according to the assumption $3 \bullet |A| < |P|$ holds. Thus step [β] finds the correct agreement. So it suffices to show that all honest participants decide for the value received in [β].

Either an honest participant P_i has received all factorizations directly from the attacker. Or in phase [n+1] of step [α] with probability at least $(1\text{-}2^{-d})^{n-1}$ at least one honest participant P_i knows all factorizations. In both cases he broadcasts them as D_i in step [γ_j]. Because of $3 \bullet |A| < |P|$ the broadcast of step [γ_j] is correct, thus all honest participants receive this D_i. So in [End], they decide for the value received in [β].

3rd case: The attacker can forge signatures, but he does not use this ability disturb step [α]. (But perhaps he disturbs the steps [γ_j] by sending around factorizations!).

Then according to the assumption $3 \bullet |A| < |P|$ holds. Thus the byzantine agreements of all steps [γ_j] are correct, so all honest participants receive the same messages D_i. Therefore either all of them decide for [α], or all of them for [β] in [End].

Also because of $3 \cdot |A| < |P|$, if they decide for [β], the value they get is correct. As the attacker has not disturbed step [α], by lemma 4.4.i, step [α] also found a correct agreement. So no matter which decision the honest participants take in [End], it is correct. □

There are some obvious efficiency improvements:

- To save messages, one can postpone step [β] until after the steps [γ_j] and only perform it if the decision is that its result must be used. (The proof showed that there will be agreement among all honest participants upon this.)
- To improve the ratio between decision making and the original agreement protocols, one can decide about the correctness of an arbitrary number of steps [α] with one execution of the steps [γ_j]. Of course, then participant P_i chooses the factorizations as D_i if he had them at the end of any of the steps [α].
- To save phases, step [β] can be executed in parallel with any of the other steps. Also all steps [γ_j] can be executed in parallel. This leaves $(n+1) + \lceil n/3 \rceil$ phases.

One can save one more phase at the expense of more messages (namely phase [n+1] of the supplement): To this end, in phase [n] each participant P_i additionally sends all the square roots which he prepared for his own signatures to all other participants. This does not disturb BA1, as that is already finished. Then each participant can perform the factorizations on his own, which he would otherwise get in phase [n+1]. Thus one has $\lceil 4/3 \cdot n \rceil$ phases.

This is also possible in section 4.4.3.3. Of course, then the signed value of the sender cannot be later used as proof towards a third party, but this is not necessary in our applications.

6 Summary

The goal of this report was to investigate how a sender and recipient untraceability scheme based on superposed sending (sect. 2.1) can be realized.

We have shown that untraceability can be realized in a really *unconditional* way, i.e. without making any assumptions about the network or the attacker, by combining **superposed sending** ([Chau_88] and sect. 2) and **fail-stop broadcast** (sect. 3.2).

This improves the result of David Chaum who implicitly assumes the existence of a reliable broadcast network.

Unfortunately the unconditional untraceability scheme doesn't guarantee **serviceability**, i.e. each dishonest participant can stop the network. Thus in section 4 we discussed how to guarantee serviceability in spite of untraceability.

The protocol for solving this problem suggested in [Chau_88] can be misused for an easy active attack on untraceability (sect. 4.2), but its basic idea can be used for the secure solutions presented in sections 4.3-4.5.

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 7.

Protocol	Section	A_{serv}	A_{untr}	Untraceability probabilistic
Superposed sending	2.1	$A = \emptyset$	reliable broadcast	no
Fail stop broadcast	3.2.2.1	$A = \emptyset$	$\emptyset$	no
	3.2.2.2 - 3.2.2.3	$A = \emptyset$	$\emptyset$	yes
		reliable broadcast assumption		
	4.3	$\wedge$ comp. restr.	$\wedge\ \emptyset$	no
Reliable broadcast	4.3	$\wedge\ 3 \cdot \lvert A \rvert < \lvert P \rvert$	$\wedge\ 3 \cdot \lvert A \rvert < \lvert P \rvert$	no
	4.5	$\wedge\ 2 \cdot \lvert A \rvert + 1 < \lvert P \rvert$	$\wedge\ 2 \cdot \lvert A \rvert + 1 < \lvert P \rvert$	yes
		attacker cannot prevent communication between honest participants		
Byzantine agreement	4.4.1.2	$\wedge\ 3 \cdot \lvert A \rvert < \lvert P \rvert$ (Untraceability probabilistic if authentication codes are necessary)		
	4.4.1.2	$\wedge$ A comp. restricted		yes
		attacker cannot prevent communication between honest participants		
Fail-stop agreement	4.4.3	$\wedge$ comp. restr.	$\wedge\ \emptyset$	yes

Figure 7 Summary

Our solution described in section 4.4.3 is based on **fail-stop Byzantine Agreement**, i.e. Byzantine agreement with signatures and the additional property that as soon as the attacker is able to forge 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 be disturbed only by an attacker who controls more than a third of all participants *and* who is able to forge signatures (sect. 5).

Acknowledgements

We are pleased to thank Andreas Pfitzmann for a vast amount of tricky ideas and clever improvements of our schemes, and we are grateful to Birgit Baum, Manfred Böttger, Axel Burandt, and Klaus Echtle for lots of valuable suggestions and the German Science Foundation (DFG) for financial support.

References

AdHu_87 L. M. Adleman, M. A. Huang: Recognizing Primes In Random Polynomial Time; 19th Symposium on Theory of Computing (STOC) 1987, ACM, New York 1987, 462-469.

AlSc_83 B. Alpern, F. B. Schneider: Key exchange Using 'Keyless Cryptography'; Information Processing Letters Vol. 16, 26 February 1983, 79-81.

AnLe_81 T. Anderson, P. A. Lee: Fault Tolerance - Principles and Practice; Prentice Hall, Englewood Cliffs, New Jersey, 1981.

BeGW_88 M. Ben-Or, S. Goldwasser, A. Wigderson: Completeness theorems for non-cryptographic fault-tolerant distributed computation; 20th Symposium on Theory of Computing (STOC) 1988, ACM, New York 1988, 1-10.

BeMi_88 M. Bellare, S. Micali: How to sign given any trapdoor function; 20th Symposium on Theory of Computing (STOC) 1988, ACM, New York 1988, 32-42.

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

Bura_88 A. Burandt: Informationstheoretisch unverkettbare Beglaubigung von Pseudonymen mit beliebigen Signatursystemen; Studienarbeit am Institut für Rechnerentwurf und Fehlertoleranz der Universität Karlsruhe, Mai 1988.

CGMA_85 B. Chor, S. Goldwasser, S. Micali, B. Awerbuch: Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract); 26th Symposium on Foundations of Computer Science (FOCS) 1985, IEEE Computer Society, 1985, 383-395.

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

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

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

ChCD1_88 D. Chaum, C. Crépeau, I. Damgård: Multiparty unconditional secure protocols; 20th Symposium on Theory of Computing (STOC) 1988, ACM, New York 1988, 11-19.

CoOS_86 D. Coppersmith, A. M. Odlzyko, R. Schroeppel: Discrete Logarithms in GF(p); Algorithmica 1, Springer-Verlag, Heidelberg, 1986, 1-15.

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

Dole_81 D. Dolev: The Byzantine Generals Strike Again; Department of Computer Science, Stanford University, Stanford, CA 94305, Report No. STAN-CS-81-846, March 1981; appeared in: Journal of Algorithms 3/1 (1982) 14-30.

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

EcNi_89 K. Echtle, A. Niedermaier: Eine senderanonyme fehlertolerante Kommunikationsstruktur; will be presented at: 4th Int. GI/ITG/GMA Conf. Fault-Tolerant Computing-Systems Baden-Baden 1989; IFB, Springer-Verlag, Berlin 1989.

FeMi_88 P. Feldman, S. Micali: Optimal algorithms for byzantine agreement; 20th Symposium on Theory of Computing (STOC) 1988, ACM, New York 1988, 148-161.

Gall_68 R. G. Gallager: Information Theory and Reliable Communication; John Wiley & Sons., New York 1968.

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 O. Goldreich, S. Micali, A. Wigderson: How to play any mental game - or - a completeness theorem for protocols with honest majority; 19th Symposium on Theory of Computing (STOC) 1987, ACM, New York 1987, 218-229.

GoM1_86 O. Goldreich, S. Micali, A. Wigderson: Proofs that Yield Nothing But their Validity and a Methology of Cryptographic Protocol Design; 27th Symposium on Foundations of Computer Science (FOCS) 1986, IEEE Computer Society, 1986, 174-187.

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

Kran_86 E. Kranakis: Primality and Cryptography; Wiley-Teubner Series in Computer Science, B. G. Teubner, Stuttgart 1986.

LaSP_82 L. Lamport, R. Shostak, M. Pease: The Byzantine Generals Problem; ACM Transactions on Programming Languages and Systems (TOPLAS) 4/3 (1982) 382-401.

LoWi_88 D. L. Long, A. Wigderson: The discrete logarithm hides O(log n) bits; SIAM J. Comput. 17/2 (1988) 363-372.

MaPf_87 A. Mann, A. Pfitzmann: Technischer Datenschutz und Fehlertoleranz in Kommunikationssystemen; Kommunikation in Verteilten Systemen, GI/NTG-Fachtagung, Aachen 1987, IFB 130, Springer-Verlag, Heidelberg 1987, 16-30; Überarbeitung in: Datenschutz und Datensicherheit DuD /8 (1987) 393-405.

Marc_88 E. Marchel: Leistungsbewertung von überlagerndem Empfangen bei Mehrfachzugriffsverfahren mittels Kollisionsauflösung; Diplomarbeit am Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, April 1988.

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

Nied_87 A. Niedermaier: Bewertung von Zuverlässigkeit und Senderanonymität einer fehlertoleranten Kommunikationsstruktur; Diplomarbeit am Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, September 1987.

Odly_85 A. M. Odlyzko: Discrete logarithms in finite fields and their cryptographic significance; Eurocrypt '84, LNCS 209, Springer-Verlag, Heidelberg 1985, 224-314.

Pera_86 R. Peralta: Simultaneous Security of Bits in the Discrete Log; Eurocrypt '85, LNCS 219, Springer-Verlag, Heidelberg 1986, 62-72.

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

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

Pfit_84 A. Pfitzmann: A switched/broadcast ISDN to decrease user observability; 1984 International Zurich Seminar on Digital Communications, Applications of Source Coding, Channel Coding and Secrecy Coding, March 6-8, 1984, Zurich, Switzerland, Proceedings IEEE Catalog no. 84CH1998-4, 183-190.

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

PfPW_88 A. Pfitzmann, B. Pfitzmann, M. Waidner: Datenschutz garantierende offene Kommunikationsnetze; Informatik-Spektrum 11/3 (1988) 118-142.

PfWa_86 A. Pfitzmann, M. Waidner: Networks without user observability -- design options; Eurocrypt '85, LNCS 219, Springer-Verlag, Heidelberg 1986, 245-253; Überarbeitung in: Computers & Security 6/2 (1987) 158-166.

Prad_86 D. K. Pradhan (ed.): Fault-tolerant Computing – Theory and Techniques; Prentice-Hall, Englewood Cliffs 1986 (2 vol.).

Rabi_79 M. O. Rabin: Digitalized Signatures and Public-Key Functions as Intractable as Factorization; Massachusetts Institute of Technology, Laboratory for Computer Science, MIT/LCS /TR-212, January 1979.

Rabi_80 M. O. Rabin: Probabilistic algorithm for primality testing; J. Number Theory 12 (1980) 128-138.

Reis_87 R. 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 G.J. Simmons: A Survey of Information Authentication; Proceedings of the IEEE 76/5 (1988) 603-620.

Tane_88 A. S. Tanenbaum: Computer Networks; 2nd ed., Prentice-Hall, Englewood Cliffs 1988.

Triv_82 K. S. Trivedi: Probability and Statistics with Reliability, Queuing, and Computer Science Applications; Prentice-Hall, Englewood Cliffs 1982.

VaVa_85 U. V. Vazirani, V. V. Vazirani: Efficient and Secure Pseudo-Random Number Generation (extended abstract); Crypto '84, LNCS 196, Springer-Verlag, Berlin 1985, 193-202.

Waid_89 M. 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.

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

Woll_87 H. Woll: Reductions among Number Theoretic Problems; Information and Computation (formerly Information and Control) 72/3 (1987) 167-179.

Update 90.06.12 to:

Unconditional Sender and Recipient Untraceability in spite of Active Attacks – Some Remarks

Interner Bericht 5/89, Universität Karlsruhe, Fakultät für Informatik, März 1989

Michael Waidner, Birgit Pfitzmann

Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe
Postfach 6980, D-7500 Karlsruhe 1, F. R. Germany
Phone: ++49-721-608-4024, Fax: ++49-721-608-4290
E-mail (CSnet): WAIDNER@IRA.UKA.DE

1. We have found a much more efficient signature scheme whose forgery can be proved in the meantime. (We now call such schemes in general "fail-stop signature schemes", the old scheme "root signature scheme" and the new scheme "hiding signature scheme".) Thus reading the root scheme (Section 4.4.3.1) and all following statements about efficiency is only of historical value.
   For descriptions and proofs of the hiding signature scheme, see [Pfit1_89, WaPf1_89, BlPW_90, PfWa_90].
2. Minor improvements to adaptive Byzantine Agreement (Chapter 5) can be found in [PfWa1_90]. Another efficient and anonymity preserving multi-access protocol (cf. Section 2.2) has been published in [BoBo_89]. Alternatives to how to understand the basic trap-protocol by Chaum (Section 4.2.1), and resulting alternatives to the improved protocol (Section 4.3), are discussed in [WaPf1_89]. (None is better than the one presented here, but some are about as good.)
3. The notation in the overview table (Figure 7) can easily be misunderstood:
   "A = ∅" means that there are no attackers. Thus it is a strong assumption.
   "∅" alone means "true", or in other words " unconditional". Thus it is the weakest possible assumption.
4. Section 4.4.3.1.3 about efficiency improvements for the root signature scheme is wrong. The lemma is correct, but both methods using it would need the stronger assumptions that all square roots of $f_m^k(a)$ are not only quadratic residues, but iterated squares. This is not true for general p.

Literature

BlPW_90 Gerrit Bleumer, Birgit Pfitzmann, Michael Waidner: A Remark on a Signature Scheme where Forgery can be Proved; Abstracts of Eurocrypt '90, Århus 1990.

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

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

PfWa_90 Birgit Pfitzmann, Michael Waidner: Fail-stop Signatures and their Application to Byzantine Agreement; submitted for publication, Universität Karlsruhe 1990.

PfWa1_90 {Birgit Pfitzmann, Michael Waidner}: More Secure Variants of Byzantine Agreement; submitted for publication, Universität Karlsruhe 1990.

WaPf1_89 Michael Waidner, Birgit Pfitzmann: The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability; complete version of an abstract presented at Eurocrypt '89; Universität Karlsruhe 1989.