RZ 2882 (#90830) 11/18/96
Computer Science/Mathematics 25 pages

Research Report

Information-Theoretic Pseudosignatures and Byzantine Agreement for $t \geq n/3$

Birgit Pfitzmann

Institut für Informatik
Universität Hildesheim
Samelsonplatz 1
D-31141 Hildesheim, Germany
email: pfitzb@informatik.uni-hildesheim.de

Michael Waidner

IBM Research Division
Zürich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon, Switzerland
email: wmi@zurich.ibm.com

Research Division
Almaden • T.J. Watson • Tokyo • Zurich

Information-Theoretic Pseudosignatures and Byzantine Agreement for $t \geq n/3$[*]

Birgit Pfitzmann[†], Michael Waidner[‡]

Abstract. Byzantine agreement means achieving reliable broadcast on a point-to-point network of n processors, of which up to t may be maliciously faulty. A well-known result by Pease, Shostak, and Lamport says that perfect Byzantine agreement is only possible if $t < n/3$. In contrast, so-called authenticated protocols achieve Byzantine agreement for any t based on computational assumptions, typically the existence of a digital signature scheme, an assumption equivalent to the existence of one-way functions. The "folklore" belief based on these two results is that computational assumptions are necessary to achieve Byzantine agreement for $t \geq n/3$.

We present a protocol that refutes this folklore belief, i.e., it achieves Byzantine agreement for any t in an information-theoretic setting. It does not, however, contradict the precise impossibility result: More than one difference exists between the model in that proof and the model of the existing authenticated protocols, and we only remove the computational assumption.

Our protocol is based on a new information-theoretically secure authentication scheme with many of the properties of digital signatures; we call it pseudosignatures. Our construction of pseudosignatures generalizes a scheme by Chaum and Roijakkers.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design; C.2.4 [Computer-Communication Networks]: Distributed Systems; C.4 [Performance Of Systems]: *reliability, availability, and serviceability*; D.4.1 [Operating Systems]: Process Management—*concurrency*; *synchronization*; D.4.5 [Operating Systems]: Reliability—*fault-tolerance*; D.4.6 [Operating Systems]: Security And Protection—*authentication*; E.3 [Data Encryption]; E.4 [Coding And Information Theory]

General Terms: Distributed computing, Reliability, Cryptology

Additional Keywords and Phrases: Byzantine agreement, reliable broadcast, signatures, fault tolerance, information-theoretic security

---

[*] An extended abstract of this paper was presented at the *9th Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 577, Springer-Verlag, Heidelberg, Germany, pp. 339-350.

[†] Institut für Informatik, Universität Hildesheim, Samelsonplatz 1, D-31141 Hildesheim, Germany, pfitzb@informatik.uni-hildesheim.de; supported by DFG (Deutsche Forschungsgemeinschaft); revision partly done at the Isaac Newton Institute, Cambridge, UK.

[‡] IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland, wmi@zurich.ibm.com; work mainly done at the Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, Germany.

IBM Research Report RZ 2882 (#90830) 11/18/96

1 Introduction

Byzantine agreement protocols are an important primitive for fault-tolerant distributed computations. They are needed to achieve reliable broadcast where this is not available physically and where some processors may be faulty. Correct agreement on a value from a transmitter means

- *consistency*. all correct processors agree on the same value v;
- *correctness*: if the transmitter is correct, v is the value it intended to send.

Let n be the number of processors and t an upper bound on the number of faulty ones. Faulty processors may behave maliciously.

Byzantine agreement was introduced in [PSL80], where it was proved that perfect Byzantine agreement is only possible if $t < n/3$. Many subsequent papers have studied upper and lower bounds on the attainable efficiency of such protocols, see [CD89, BMD93] for overviews.

However, if digital signatures [DH76, GMR88] can be used, Byzantine agreement is possible for any $t < n$, even in polynomial time [DS83]. Such protocols are called authenticated protocols. Digital signatures exist if and only if one-way functions exist, which is an unproven computational assumption that implies P≠NP [R90]. This has led to the “folklore” that Byzantine agreement for $t \geq n/3$ necessarily relies on computational assumptions. However, to be more precise, the model used in authenticated protocols is weaker than that in the impossibility proof in [PSL80] in three ways:

- *Computational assumptions*: This means both the restriction of faulty processors to polynomial-time computations and an unproven assumption that certain problems are not computable in polynomial time. In contrast, a protocol in which the computational power of faulty processors is not restricted is called information-theoretic.
- *Broadcast in a precomputation phase* is needed to distribute the public keys required to verify the digital signatures. (In practical applications of signatures, such broadcast precomputation is typically approximated by distribution with certification by a third party. However, such an approach is not possible in a Byzantine setting, in which no party is more trusted and reliable than others.)
- *A small error probability* has to be allowed, because even if the computational assumptions hold and the faulty processors only use polynomial time, they have a small chance of guessing the signature of another processor correctly.

Our main result is that the first of these three restrictions is not necessary, i.e., we present an information-theoretic Byzantine agreement protocol for any t, but, as in normal authenticated protocols, needing broadcast in a precomputation phase and with a small error probability. For a security parameter σ, the error probability is at most $2^{-\sigma}$, whereas all computations required from the correct processors are polynomial in σ. Furthermore, secret channels are needed in the precomputation phase. Details about the models and the necessity of the other restrictions are given in Sections 1.1 and 1.2, and our main construction is presented in Section 4. In Section 5, the construction is modified so that the broadcast channel is only needed during a precomputation phase of a fixed length for an arbitrary number of broadcasts.

The main primitive used in our constructions is interesting in its own right. It is a scheme that achieves message authentication with many, but not all of the properties of digital signature schemes, without relying on computational assumptions. We call it pseudosignatures. We present properties and types of authentication schemes in Section 2 and the concrete construction of a pseudosignature scheme in Section 3. This construction extends and improves the scheme in [CR91].

1.1 Models of Byzantine Agreement

Recall that n is the number of processors and σ a security parameter so that an error probability of $2^{-\sigma}$ is acceptable. Let $l \in \mathbb{N}$ be the length of the values to be agreed upon. Our own protocols are polynomial in n, l, and σ.

The processors are denoted by $P_1, \ldots, P_n$. Faulty processors are *malicious*, i.e., they can behave arbitrarily. We consider *synchronous* protocols, i.e., they proceed in rounds of fixed length. More benign fault models and less benign models of time are also known, but we do not refer to them in the following. As in the original model of [PSL80], we assume that an *authentic* point-to-point channel is available between any pair of processors.

A *precomputation* phase is a phase before the transmitter knows the value v it wants to send. We call the phase after v is known the *main* phase of the agreement protocol. Protocols with a precomputation phase typically assume that more types of channels are available in this phase than in the main phase.

Needing *broadcast* in a precomputation phase is, in our eyes, the major remaining restriction of the model after the computational assumption has been removed, i.e., the main price to pay for allowing $t \geq n/3$. It depends on the application whether one can implement it or has to hope that $t < n/3$. In some applications, however, it is quite realistic. The Byzantine generals of the original example in [LSP82] could carry out a precomputation phase while they are together in a tent before the campaign and thus have oral broadcast communication available. In a distributed system in practice, say an aircraft control system, a trustworthy operator may initially set up the processors using broadcast channels that will no longer be available in the operational environment. Note also that in the real world some kind of precomputation with reliable broadcast is always necessary (although this does not count in the model) because the processors must at least be equipped with consistent programs for the agreement protocol.

In the precomputation phase, we additionally need *secret* channels between each pair of processors, i.e., channels that are secure from eavesdropping by other processors. In practice, this seems a smaller problem than broadcast in the precomputation phase. Secret channels are not used in typical authenticated protocols, but typical concrete computational assumptions imply them: the existing authentic channels could be used for the exchange of a secret key, see [DH76, RSA78] for well-known, but not provably secure examples. However, secret key exchange is not known to exist under equally weak conditions as digital signatures [IR89].

1.2 Optimality of Our Result

We now show that our model is optimal for information-theoretic Byzantine agreement with $t \geq n/3$, i.e., it is not possible to eliminate any of the remaining restrictions.

- Error probability: First, one can see that the proof in [PSL80], primarily intended for information-theoretic deterministic protocols without precomputation, holds for any deterministic protocol, even if broadcast and secret channels are available in a precomputation phase. Consequently, it also holds for all error-free protocols (where we count a failure to terminate in bounded time as an error): If a probabilistic protocol is error-free, any deterministic protocol derived from it by fixing all the random choices to, say, 0 is also correct.

- Broadcast in the precomputation phase: It is sketched in [DD91] that even if secret channels are given a protocol without broadcast in a precomputation phase has an error probability of at least 1/3, which is not a small error probability. The best published result is [GY89], building upon unpublished work by Karlin and Yao; it shows a slightly larger error probability if no secret channels are given.
- Secret channels in the precomputation phase: We show that the result of [GY89] can be used even if there is broadcast in a precomputation phase. Let *pub* denote the entire data sent during an execution of the precomputation. As *pub* is known to all processors, we can regard the remaining protocol, say, *main_prot*(*pub*), as a protocol without precomputation: If the correct processors made random choices in the precomputation, they now have to make corresponding choices with the conditional probabilities induced by *pub*. This is possible because the result in [GY89] does not rely on restrictions on the computational abilities of the correct processors. By [GY89], the remaining protocol therefore has an error probability of at least 1/3. Averaging over *pub* yields that also the complete protocol has an error probability of at least 1/3.

We presented the first information-theoretic Byzantine agreement protocol for $t \geq n/3$ in [BPW91]. The model is the same as here, but the number of faulty processors was still restricted to $t < n/2$. This result was extended to a protocol with an expected constant number of rounds in [W91].

2 Properties and Types of Authentication Schemes

In this section, we present a framework for the definition and classification of different types of authentication schemes. This is useful for a clear definition of pseudosignatures and for discussing in detail why previous types of authentication schemes could not be used to provide information-theoretic Byzantine agreement. The framework is an extension of one for different types of signature schemes described in [P93, P96]. It may look unfamiliar because it primarily considers the interface behavior of the authentication systems, not internal objects such as keys and signatures. This approach is useful for a unified treatment of systems that, e.g., have interactive protocols where others simply have a key or a signature. Moreover, it allows us to define what it means for one property to be fulfilled with different degrees of security, whereas usual cryptographic definitions have no formal notion of a "property" independent of its degree of security.

2.1 Minimal Properties of Authentication

We define authentication systems as distributed reactive systems that have at least two types of processors, called originators' and recipients' processors, and offer at least two types of transactions, namely *initialization* and *authentication*. A transaction is a subprotocol carried out among a subset of the processors. In particular, authentication is usually carried out between one originator's processor and one recipient's processor. Note that we reserve the terms originator and recipient for the *users* of such a system. We use "she" and "he" for an originator Alice and a recipient Bob, as usual in cryptology.

An authentication *scheme* consists of the finite description of such systems, primarily the programs for the two types of processors.[1] A scheme can be instantiated into a system with different *system parameters*. These are the number of processors of each type and the initial states of these processors. The initial state typically consists of the security parameters such as σ and the processor's own identity (number).

Initialization is a transaction between at least one originator and one or more recipients. The originator inputs (*'init'*, ids_R, N) to her processor, where *'init'* is a string indicating the type of transaction wanted, ids_R designates the desired potential recipients, and N is an upper bound on the number of messages to be authenticated later. Both ids_R and N have default values for "no restriction". Each recipient inputs (*'init'*, id_S, N) to his processor, where id_S is the identity of the originator. Each processor outputs (acc, $ids_{R,out}$), where acc is a Boolean value that denotes whether initialization was successful at all and $ids_{R,out}$ can be used to indicate which recipients took part successfully. Internally, the processors carry out key generation and distribution for this originator and the recipients that participate.

For authentication, the originator inputs (*'auth'*, msg), where msg is the message.[2] The recipient inputs (*'test'*, msg, id_S), where id_S is the identity of the presumed originator. The recipient's processor outputs a Boolean value acc that denotes whether the message was successfully authenticated. Internally, the two processors may carry out any type of protocol.

The transactions may have more parameters and there may be more transactions as long as they do not interfere with the following minimal requirements imposed on every authentication scheme:

- Effectiveness of initialization: If the originator and at least one recipient are honest (i.e., they make consistent inputs, and their processors use the prescribed programs), their result in initialization is $acc = true$.
- Correctness of initialization: All correct processors make the same output (acc, $ids_{R,out}$), and the identities of all correct recipients' processors are in $ids_{R,out}$.
- Effectiveness of authentication: If an originator and a recipient who have successfully carried out initialization (usually with more recipients) make consistent inputs for an authentication and the number of previous authentications with respect to this initialization is less than N, the result is *true*.
- Unforgeability: If an originator has never authenticated a certain message msg, i.e., input (*'auth'*, msg), no honest recipient will believe that she did, that is, on input (*'test'*, msg, id_S), a correct recipient's processor will output $acc = false$.

The requirements can be formulated in detail in any language for specifying sequences of events, e.g., temporal logic. (The details are mainly the exact formulations of consistent inputs and previous successful initialization.)

---

[1] The word "protocol" is used in the same way in other contexts. We sometimes use it for schemes with only one or two transactions, such as Byzantine agreement, and for the part of a scheme corresponding to one transaction.

[2] If several initializations have been made with one processor, an authentication must refer to a particular one. The easiest way to do this is to recognize that what is called a processor here is a software entity, of which there may be many in a hardware device, and to require that new entities are instantiated for each initialization [P93]. Otherwise, one needs initialization identifiers.

[P93, P96] sketch what it means for such requirements to be fulfilled under active attacks and in an information-theoretic or computational sense, i.e., several types of cryptologic semantics of temporal requirements on in- and outputs. The aspects we need are presented in more detail in Section 2.4.

2.2 Transferability

Byzantine agreement protocols such as [DS83] use an additional transaction *transfer* that allows one recipient to transfer the authentication of a message to another recipient. The first recipient inputs (*'transfer'*, *msg*, id_S), the second one (*'test'*, *msg*, id_S). The second recipient obtains a Boolean output *acc*. If such a scheme also allows initialization with an arbitrary number of recipients, i.e., ids_R = "no restriction", we call it an *authentication scheme with arbitrary transferability*. The following requirement is made:

- Transferability: If a recipient has obtained an authenticated message, i.e., input (*'test'*, *msg*, id_S) and obtained the output *acc* = *true*, he can transfer it to any other honest recipient, i.e., if they make consistent inputs for a transfer, the second recipient will also obtain the output *acc* = *true*.

2.3 Existing Authentication Schemes in this Framework

Digital signature schemes [DH76, GMR88] are authentication schemes with arbitrary transferability and a particularly simple structure: All transactions are noninteractive. In initialization, the originator's processor executes a probabilistic algorithm *gen* to generate a pair (*sk*, *pk*) of a secret and a public key, and broadcasts *pk* to the other processors. In authentication, the originator's processor carries out a (probabilistic) algorithm *sign*(*sk*, *msg*) and sends the result *s*, called signature, to the recipient's processor, which tests it using a deterministic algorithm *test*(*pk*, *msg*, *s*). In a transfer, the first recipient's processor simply forwards *s*, and the second recipient's processor tests it with *test*(*pk*, *msg*, *s*) like the first one.

With this structure, effectiveness and correctness of initialization and arbitrary transferability are trivially fulfilled (error-free), but unforgeability can only be fulfilled computationally. For instance, an NP-easy[3] forging algorithm is that a forger guesses a signature s^* and verifies this guess with *test*(*pk*, *msg*, s^*). More precisely, the existence of digital signatures in this sense implies the existence of a one-way function [R90].

Authenticated Byzantine agreement protocols like [DS83] are formulated in terms of digital signatures in this sense; at least they assume that signing and transfers are noninteractive. However, one can quite easily see that any authentication scheme with arbitrary transferability could be used. (Of course, the number of rounds of the protocols would increase accordingly). Thus any such scheme in which all five requirements were fulfilled information-theoretically would easily yield information-theoretic Byzantine agreement for any *t*.

Let us now investigate known authentication schemes with information-theoretic security.

So-called *authentication codes* have been known for a long time [GMS74, WC81]. In the following, we will need them as subprotocols. Initialization usually is only for one recipient, i.e., ids_R is this recipient's identity. Either of the originator's and the recipient's processor executes a probabilistic algorithm *gen* to generate a key *ak* and sends it to the other processor.

---

[3] Roughly, this means NP for search problems; see [GJ79].

In authentication, the originator's processor carries out an algorithm *auth*(*ak*, *msg*) and sends the result *mac*, called message authentication code, to the recipient's processor, which tests it by also computing *auth*(*ak*, *msg*) and comparing it with *mac*.[4] As initialization is for only one recipient, no transaction *transfer* is possible. (If more than two parties share a key *ak*, unforgeability cannot be guaranteed for any two of them.) Thus authentication codes only implement authentic channels and are no help for Byzantine agreement.

In *arbitrated authentication codes*, as introduced in [S90], one specific third party, called arbiter, takes part in initialization with one originator and one recipient, and the recipient can later convince this arbiter of an authentication. This is a kind of transfer to one specific party. Most of these schemes only guarantee unforgeability and the other three properties if the arbiter is correct, and are therefore not applicable in a Byzantine scenario. In [BS88], multiple arbiters jointly decide whether a message was authenticated, and a dishonest minority of them cannot forge. So-called schemes with security against the arbiter [DY91, JS95] eliminate the problem of forgeability, and in [DY90] it is also shown how effectiveness of authentication can be guaranteed even if the arbiter is faulty (in initialization). However, still only one transfer is possible, and thus the scheme cannot be used in known authenticated Byzantine agreement protocols for $t > 1$. Actually, [DD91] sketches that any authentication mechanism that does not guarantee at least t transfers is insufficient for any Byzantine agreement protocol with malicious faults.

An interesting related scheme is the *check vector* scheme of [R94]. It has an originator, one first and one or more second recipients. The first recipient can transfer a value received from the originator to any of the second recipients. However, it is not really an authentication scheme, because initialization and authentication are joined: the originator already has to know the value to be authenticated and tell it to the first recipient in this transaction. Moreover, this transaction needs a broadcast channel if no party is trusted (see "Modified Verification of Check Vectors" in Figure 3 of [R94]). This makes the scheme unsuitable for Byzantine agreement protocols, in which the values to be authenticated are only known in the main phase.

2.4 Definition of Pseudosignatures

Pseudosignatures are slightly weaker than authentication schemes with arbitrary transferability. The two restrictions are that transferability is only finite and that active attacks on recipients must be restricted. This allows us to fulfil all requirements information-theoretically.

2.4.1 Finite Transferability

Finite transferability means that the certainty about the authenticity of a message may decrease with each transfer, so that only a finite number λ of successive transfers is guaranteed. The parameter λ can be chosen as an input in initialization. The restriction only applies to transfers in series; the number of transfers by the same recipient is not restricted. The inputs of the first and the second recipient in the λ^*-th successive transfer of a message are (*'transfer'*, *msg*, id_S, λ^*) and (*'test'*, *msg*, id_S, λ^*). If the output is *true*, we say that the second recipient has λ^*-accepted the message. The command (*'test'*, *msg*, id_S) from authentication could now be called (*'test'*, *msg*, id_S, 0).

---

[4] In rare cases, *auth* is probabilistic, and the recipient uses an algorithm *test*(*ak*, *msg*, *mac*).

- Finite transferability: If a recipient has λ'-accepted a message for $\lambda' < \lambda$, he can transfer it for any λ'' with $\lambda' < \lambda'' \leq \lambda$. That is, if he inputs (*'transfer'*, *msg*, id_S, λ'') and the second recipient inputs (*'test'*, *msg*, id_S, λ''), the second recipient also obtains the output *acc* = *true*.

2.4.2 Active Attacks

Restricted active attacks on recipients may sound strange at first, because with ordinary digital signature schemes, active attacks are only considered against originators [GMR88]. However, in a more general authentication scheme, in which recipients' processors also have secrets, active attacks against them must also be considered. Generally, in the definition of fulfilling temporal requirements on in- and outputs in a cryptologic sense, it is assumed that the honest users, i.e., the entities that make the inputs to and receive the outputs from the correct processors, are arbitrarily influenced by the attackers, e.g., they may authenticate messages chosen by the attacker or tell the attacker the results of tests. This is shown in Figure 1.

Note that this scenario in particular prescribes that honest users only use the correct processors by the offered inputs and outputs, and not, e.g., by reading out the processors' internal secrets.

Research Report

nformation-Theoretic Pseudosignatures and Byzantine Agreement for $t \geq n/3$

Birgit Pfitzmann

nstitut für Informatik
Universität Hildesheim
Samelsonplatz 1
D-31141 Hildesheim, Germany
email: pfitzb@informatik.uni-hildesheim.de

Michael Waidner

BM Research Division
Zürich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon, Switzerland
email: wmi@zurich.ibm.com

Research Division
Almaden • T.J. Watson • Tokyo • Zurich

IBM Research Report RZ 2882 (#90830) 11/18/96

Information-Theoretic Pseudosignatures and Byzantine Agreement for $t \geq n/3$[*]

Birgit Pfitzmann[†], Michael Waidner[‡]

Abstract. Byzantine agreement means achieving reliable broadcast on a point-to-point network of n processors, of which up to t may be maliciously faulty. A well-known result by Pease, Shostak, and Lamport says that perfect Byzantine agreement is only possible if $t < n/3$. In contrast, so-called authenticated protocols achieve Byzantine agreement for any t based on computational assumptions, typically the existence of a digital signature scheme, an assumption equivalent to the existence of one-way functions. The "folklore" belief based on these two results is that computational assumptions are necessary to achieve Byzantine agreement for $t \geq n/3$.

We present a protocol that refutes this folklore belief, i.e., it achieves Byzantine agreement for any t in an information-theoretic setting. It does not, however, contradict the precise impossibility result: More than one difference exists between the model in that proof and the model of the existing authenticated protocols, and we only remove the computational assumption.

Our protocol is based on a new information-theoretically secure authentication scheme with many of the properties of digital signatures; we call it pseudosignatures. Our construction of pseudosignatures generalizes a scheme by Chaum and Roijakkers.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design; C.2.4 [Computer-Communication Networks]: Distributed Systems; C.4 [Performance Of Systems]: *reliability, availability, and serviceability*; D.4.1 [Operating Systems]: Process Management—*concurrency*; *synchronization*; D.4.5 [Operating Systems]: Reliability—*fault-tolerance*; D.4.6 [Operating Systems]: Security And Protection—*authentication*; E.3 [Data Encryption]; E.4 [Coding And Information Theory]

General Terms: Distributed computing, Reliability, Cryptology

Additional Keywords and Phrases: Byzantine agreement, reliable broadcast, signatures, fault tolerance, information-theoretic security

---

[*] An extended abstract of this paper was presented at the *9th Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 577, Springer-Verlag, Heidelberg, Germany, pp. 339-350.

[†] Institut für Informatik, Universität Hildesheim, Samelsonplatz 1, D-31141 Hildesheim, Germany, pfitzb@informatik.uni-hildesheim.de; supported by DFG (Deutsche Forschungsgemeinschaft); revision partly done at the Isaac Newton Institute, Cambridge, UK.

[‡] IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland, wmi@zurich.ibm.com; work mainly done at the Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, Germany.

IBM Research Report RZ 2882 (#90830) 11/18/96

1 Introduction

Byzantine agreement protocols are an important primitive for fault-tolerant distributed computations. They are needed to achieve reliable broadcast where this is not available physically and where some processors may be faulty. Correct agreement on a value from a transmitter means

- *consistency*. all correct processors agree on the same value v;
- *correctness*: if the transmitter is correct, v is the value it intended to send.

Let n be the number of processors and t an upper bound on the number of faulty ones. Faulty processors may behave maliciously.

Byzantine agreement was introduced in [PSL80], where it was proved that perfect Byzantine agreement is only possible if $t < n/3$. Many subsequent papers have studied upper and lower bounds on the attainable efficiency of such protocols, see [CD89, BMD93] for overviews.

However, if digital signatures [DH76, GMR88] can be used, Byzantine agreement is possible for any $t < n$, even in polynomial time [DS83]. Such protocols are called authenticated protocols. Digital signatures exist if and only if one-way functions exist, which is an unproven computational assumption that implies P≠NP [R90]. This has led to the “folklore” that Byzantine agreement for $t \geq n/3$ necessarily relies on computational assumptions. However, to be more precise, the model used in authenticated protocols is weaker than that in the impossibility proof in [PSL80] in three ways:

- *Computational assumptions*: This means both the restriction of faulty processors to polynomial-time computations and an unproven assumption that certain problems are not computable in polynomial time. In contrast, a protocol in which the computational power of faulty processors is not restricted is called information-theoretic.
- *Broadcast in a precomputation phase* is needed to distribute the public keys required to verify the digital signatures. (In practical applications of signatures, such broadcast precomputation is typically approximated by distribution with certification by a third party. However, such an approach is not possible in a Byzantine setting, in which no party is more trusted and reliable than others.)
- *A small error probability* has to be allowed, because even if the computational assumptions hold and the faulty processors only use polynomial time, they have a small chance of guessing the signature of another processor correctly.

Our main result is that the first of these three restrictions is not necessary, i.e., we present an information-theoretic Byzantine agreement protocol for any t, but, as in normal authenticated protocols, needing broadcast in a precomputation phase and with a small error probability. For a security parameter σ, the error probability is at most $2^{-\sigma}$, whereas all computations required from the correct processors are polynomial in σ. Furthermore, secret channels are needed in the precomputation phase. Details about the models and the necessity of the other restrictions are given in Sections 1.1 and 1.2, and our main construction is presented in Section 4. In Section 5, the construction is modified so that the broadcast channel is only needed during a precomputation phase of a fixed length for an arbitrary number of broadcasts.

The main primitive used in our constructions is interesting in its own right. It is a scheme that achieves message authentication with many, but not all of the properties of digital signature schemes, without relying on computational assumptions. We call it pseudosignatures. We present properties and types of authentication schemes in Section 2 and the concrete construction of a pseudosignature scheme in Section 3. This construction extends and improves the scheme in [CR91].

1.1 Models of Byzantine Agreement

Recall that n is the number of processors and σ a security parameter so that an error probability of $2^{-\sigma}$ is acceptable. Let $l \in \mathbb{N}$ be the length of the values to be agreed upon. Our own protocols are polynomial in n, l, and σ.

The processors are denoted by $P_1, \ldots, P_n$. Faulty processors are *malicious*, i.e., they can behave arbitrarily. We consider *synchronous* protocols, i.e., they proceed in rounds of fixed length. More benign fault models and less benign models of time are also known, but we do not refer to them in the following. As in the original model of [PSL80], we assume that an *authentic* point-to-point channel is available between any pair of processors.

A *precomputation* phase is a phase before the transmitter knows the value v it wants to send. We call the phase after v is known the *main* phase of the agreement protocol. Protocols with a precomputation phase typically assume that more types of channels are available in this phase than in the main phase.

Needing *broadcast* in a precomputation phase is, in our eyes, the major remaining restriction of the model after the computational assumption has been removed, i.e., the main price to pay for allowing $t \geq n/3$. It depends on the application whether one can implement it or has to hope that $t < n/3$. In some applications, however, it is quite realistic. The Byzantine generals of the original example in [LSP82] could carry out a precomputation phase while they are together in a tent before the campaign and thus have oral broadcast communication available. In a distributed system in practice, say an aircraft control system, a trustworthy operator may initially set up the processors using broadcast channels that will no longer be available in the operational environment. Note also that in the real world some kind of precomputation with reliable broadcast is always necessary (although this does not count in the model) because the processors must at least be equipped with consistent programs for the agreement protocol.

In the precomputation phase, we additionally need *secret* channels between each pair of processors, i.e., channels that are secure from eavesdropping by other processors. In practice, this seems a smaller problem than broadcast in the precomputation phase. Secret channels are not used in typical authenticated protocols, but typical concrete computational assumptions imply them: the existing authentic channels could be used for the exchange of a secret key, see [DH76, RSA78] for well-known, but not provably secure examples. However, secret key exchange is not known to exist under equally weak conditions as digital signatures [IR89].

1.2 Optimality of Our Result

We now show that our model is optimal for information-theoretic Byzantine agreement with $t \geq n/3$, i.e., it is not possible to eliminate any of the remaining restrictions.

- Error probability: First, one can see that the proof in [PSL80], primarily intended for information-theoretic deterministic protocols without precomputation, holds for any deterministic protocol, even if broadcast and secret channels are available in a precomputation phase. Consequently, it also holds for all error-free protocols (where we count a failure to terminate in bounded time as an error): If a probabilistic protocol is error-free, any deterministic protocol derived from it by fixing all the random choices to, say, 0 is also correct.

- Broadcast in the precomputation phase: It is sketched in [DD91] that even if secret channels are given a protocol without broadcast in a precomputation phase has an error probability of at least 1/3, which is not a small error probability. The best published result is [GY89], building upon unpublished work by Karlin and Yao; it shows a slightly larger error probability if no secret channels are given.
- Secret channels in the precomputation phase: We show that the result of [GY89] can be used even if there is broadcast in a precomputation phase. Let *pub* denote the entire data sent during an execution of the precomputation. As *pub* is known to all processors, we can regard the remaining protocol, say, *main_prot*(*pub*), as a protocol without precomputation: If the correct processors made random choices in the precomputation, they now have to make corresponding choices with the conditional probabilities induced by *pub*. This is possible because the result in [GY89] does not rely on restrictions on the computational abilities of the correct processors. By [GY89], the remaining protocol therefore has an error probability of at least 1/3. Averaging over *pub* yields that also the complete protocol has an error probability of at least 1/3.

We presented the first information-theoretic Byzantine agreement protocol for $t \geq n/3$ in [BPW91]. The model is the same as here, but the number of faulty processors was still restricted to $t < n/2$. This result was extended to a protocol with an expected constant number of rounds in [W91].

2 Properties and Types of Authentication Schemes

In this section, we present a framework for the definition and classification of different types of authentication schemes. This is useful for a clear definition of pseudosignatures and for discussing in detail why previous types of authentication schemes could not be used to provide information-theoretic Byzantine agreement. The framework is an extension of one for different types of signature schemes described in [P93, P96]. It may look unfamiliar because it primarily considers the interface behavior of the authentication systems, not internal objects such as keys and signatures. This approach is useful for a unified treatment of systems that, e.g., have interactive protocols where others simply have a key or a signature. Moreover, it allows us to define what it means for one property to be fulfilled with different degrees of security, whereas usual cryptographic definitions have no formal notion of a “property” independent of its degree of security.

2.1 Minimal Properties of Authentication

We define authentication systems as distributed reactive systems that have at least two types of processors, called originators’ and recipients’ processors, and offer at least two types of transactions, namely *initialization* and *authentication*. A transaction is a subprotocol carried out among a subset of the processors. In particular, authentication is usually carried out between one originator’s processor and one recipient’s processor. Note that we reserve the terms originator and recipient for the *users* of such a system. We use “she” and “he” for an originator Alice and a recipient Bob, as usual in cryptology.

An authentication *scheme* consists of the finite description of such systems, primarily the programs for the two types of processors.[1] A scheme can be instantiated into a system with different *system parameters*. These are the number of processors of each type and the initial states of these processors. The initial state typically consists of the security parameters such as σ and the processor's own identity (number).

Initialization is a transaction between at least one originator and one or more recipients. The originator inputs (*'init'*, ids_R, N) to her processor, where *'init'* is a string indicating the type of transaction wanted, ids_R designates the desired potential recipients, and N is an upper bound on the number of messages to be authenticated later. Both ids_R and N have default values for "no restriction". Each recipient inputs (*'init'*, id_S, N) to his processor, where id_S is the identity of the originator. Each processor outputs (acc, $ids_{R,out}$), where acc is a Boolean value that denotes whether initialization was successful at all and $ids_{R,out}$ can be used to indicate which recipients took part successfully. Internally, the processors carry out key generation and distribution for this originator and the recipients that participate.

For authentication, the originator inputs (*'auth'*, msg), where msg is the message.[2] The recipient inputs (*'test'*, msg, id_S), where id_S is the identity of the presumed originator. The recipient's processor outputs a Boolean value acc that denotes whether the message was successfully authenticated. Internally, the two processors may carry out any type of protocol.

The transactions may have more parameters and there may be more transactions as long as they do not interfere with the following minimal requirements imposed on every authentication scheme:

- Effectiveness of initialization: If the originator and at least one recipient are honest (i.e., they make consistent inputs, and their processors use the prescribed programs), their result in initialization is $acc = true$.
- Correctness of initialization: All correct processors make the same output (acc, $ids_{R,out}$), and the identities of all correct recipients' processors are in $ids_{R,out}$.
- Effectiveness of authentication: If an originator and a recipient who have successfully carried out initialization (usually with more recipients) make consistent inputs for an authentication and the number of previous authentications with respect to this initialization is less than N, the result is *true*.
- Unforgeability: If an originator has never authenticated a certain message msg, i.e., input (*'auth'*, msg), no honest recipient will believe that she did, that is, on input (*'test'*, msg, id_S), a correct recipient's processor will output $acc = false$.

The requirements can be formulated in detail in any language for specifying sequences of events, e.g., temporal logic. (The details are mainly the exact formulations of consistent inputs and previous successful initialization.)

---

[1] The word "protocol" is used in the same way in other contexts. We sometimes use it for schemes with only one or two transactions, such as Byzantine agreement, and for the part of a scheme corresponding to one transaction.

[2] If several initializations have been made with one processor, an authentication must refer to a particular one. The easiest way to do this is to recognize that what is called a processor here is a software entity, of which there may be many in a hardware device, and to require that new entities are instantiated for each initialization [P93]. Otherwise, one needs initialization identifiers.

[P93, P96] sketch what it means for such requirements to be fulfilled under active attacks and in an information-theoretic or computational sense, i.e., several types of cryptologic semantics of temporal requirements on in- and outputs. The aspects we need are presented in more detail in Section 2.4.

2.2 Transferability

Byzantine agreement protocols such as [DS83] use an additional transaction *transfer* that allows one recipient to transfer the authentication of a message to another recipient. The first recipient inputs (*'transfer'*, *msg*, id_S), the second one (*'test'*, *msg*, id_S). The second recipient obtains a Boolean output *acc*. If such a scheme also allows initialization with an arbitrary number of recipients, i.e., ids_R = "no restriction", we call it an *authentication scheme with arbitrary transferability*. The following requirement is made:

- Transferability: If a recipient has obtained an authenticated message, i.e., input (*'test'*, *msg*, id_S) and obtained the output *acc* = *true*, he can transfer it to any other honest recipient, i.e., if they make consistent inputs for a transfer, the second recipient will also obtain the output *acc* = *true*.

2.3 Existing Authentication Schemes in this Framework

Digital signature schemes [DH76, GMR88] are authentication schemes with arbitrary transferability and a particularly simple structure: All transactions are noninteractive. In initialization, the originator's processor executes a probabilistic algorithm *gen* to generate a pair (*sk*, *pk*) of a secret and a public key, and broadcasts *pk* to the other processors. In authentication, the originator's processor carries out a (probabilistic) algorithm *sign*(*sk*, *msg*) and sends the result *s*, called signature, to the recipient's processor, which tests it using a deterministic algorithm *test*(*pk*, *msg*, *s*). In a transfer, the first recipient's processor simply forwards *s*, and the second recipient's processor tests it with *test*(*pk*, *msg*, *s*) like the first one.

With this structure, effectiveness and correctness of initialization and arbitrary transferability are trivially fulfilled (error-free), but unforgeability can only be fulfilled computationally. For instance, an NP-easy[3] forging algorithm is that a forger guesses a signature s^* and verifies this guess with *test*(*pk*, *msg*, s^*). More precisely, the existence of digital signatures in this sense implies the existence of a one-way function [R90].

Authenticated Byzantine agreement protocols like [DS83] are formulated in terms of digital signatures in this sense; at least they assume that signing and transfers are noninteractive. However, one can quite easily see that any authentication scheme with arbitrary transferability could be used. (Of course, the number of rounds of the protocols would increase accordingly). Thus any such scheme in which all five requirements were fulfilled information-theoretically would easily yield information-theoretic Byzantine agreement for any *t*.

Let us now investigate known authentication schemes with information-theoretic security.

So-called *authentication codes* have been known for a long time [GMS74, WC81]. In the following, we will need them as subprotocols. Initialization usually is only for one recipient, i.e., ids_R is this recipient's identity. Either of the originator's and the recipient's processor executes a probabilistic algorithm *gen* to generate a key *ak* and sends it to the other processor.

---

[3] Roughly, this means NP for search problems; see [GJ79].

In authentication, the originator's processor carries out an algorithm *auth*(*ak*, *msg*) and sends the result *mac*, called message authentication code, to the recipient's processor, which tests it by also computing *auth*(*ak*, *msg*) and comparing it with *mac*.[4] As initialization is for only one recipient, no transaction *transfer* is possible. (If more than two parties share a key *ak*, unforgeability cannot be guaranteed for any two of them.) Thus authentication codes only implement authentic channels and are no help for Byzantine agreement.

In *arbitrated authentication codes*, as introduced in [S90], one specific third party, called arbiter, takes part in initialization with one originator and one recipient, and the recipient can later convince this arbiter of an authentication. This is a kind of transfer to one specific party. Most of these schemes only guarantee unforgeability and the other three properties if the arbiter is correct, and are therefore not applicable in a Byzantine scenario. In [BS88], multiple arbiters jointly decide whether a message was authenticated, and a dishonest minority of them cannot forge. So-called schemes with security against the arbiter [DY91, JS95] eliminate the problem of forgeability, and in [DY90] it is also shown how effectiveness of authentication can be guaranteed even if the arbiter is faulty (in initialization). However, still only one transfer is possible, and thus the scheme cannot be used in known authenticated Byzantine agreement protocols for $t > 1$. Actually, [DD91] sketches that any authentication mechanism that does not guarantee at least t transfers is insufficient for any Byzantine agreement protocol with malicious faults.

An interesting related scheme is the *check vector* scheme of [R94]. It has an originator, one first and one or more second recipients. The first recipient can transfer a value received from the originator to any of the second recipients. However, it is not really an authentication scheme, because initialization and authentication are joined: the originator already has to know the value to be authenticated and tell it to the first recipient in this transaction. Moreover, this transaction needs a broadcast channel if no party is trusted (see "Modified Verification of Check Vectors" in Figure 3 of [R94]). This makes the scheme unsuitable for Byzantine agreement protocols, in which the values to be authenticated are only known in the main phase.

2.4 Definition of Pseudosignatures

Pseudosignatures are slightly weaker than authentication schemes with arbitrary transferability. The two restrictions are that transferability is only finite and that active attacks on recipients must be restricted. This allows us to fulfil all requirements information-theoretically.

2.4.1 Finite Transferability

Finite transferability means that the certainty about the authenticity of a message may decrease with each transfer, so that only a finite number λ of successive transfers is guaranteed. The parameter λ can be chosen as an input in initialization. The restriction only applies to transfers in series; the number of transfers by the same recipient is not restricted. The inputs of the first and the second recipient in the λ^*-th successive transfer of a message are (*'transfer'*, *msg*, id_S, λ^*) and (*'test'*, *msg*, id_S, λ^*). If the output is *true*, we say that the second recipient has λ^*-accepted the message. The command (*'test'*, *msg*, id_S) from authentication could now be called (*'test'*, *msg*, id_S, 0).

---

[4] In rare cases, *auth* is probabilistic, and the recipient uses an algorithm *test*(*ak*, *msg*, *mac*).

- Finite transferability: If a recipient has λ'-accepted a message for $\lambda' < \lambda$, he can transfer it for any λ'' with $\lambda' < \lambda'' \leq \lambda$. That is, if he inputs (*'transfer'*, *msg*, id_S, λ'') and the second recipient inputs (*'test'*, *msg*, id_S, λ''), the second recipient also obtains the output *acc* = *true*.

2.4.2 Active Attacks

Restricted active attacks on recipients may sound strange at first, because with ordinary digital signature schemes, active attacks are only considered against originators [GMR88]. However, in a more general authentication scheme, in which recipients' processors also have secrets, active attacks against them must also be considered. Generally, in the definition of fulfilling temporal requirements on in- and outputs in a cryptologic sense, it is assumed that the honest users, i.e., the entities that make the inputs to and receive the outputs from the correct processors, are arbitrarily influenced by the attackers, e.g., they may authenticate messages chosen by the attacker or tell the attacker the results of tests. This is shown in Figure 1.

Note that this scenario in particular prescribes that honest users only use the correct processors by the offered inputs and outputs, and not, e.g., by reading out the processors' internal secrets.

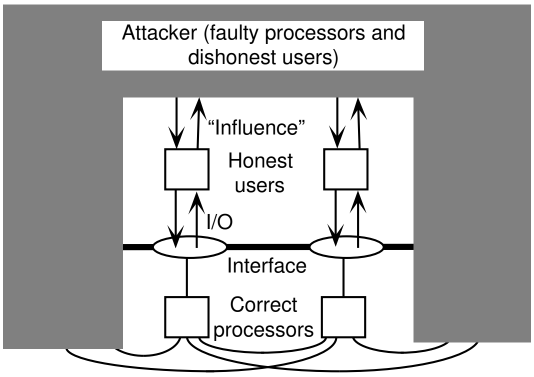

Figure 1 General active attack on two correct processors. For instance, they might be the processors of two recipients in a transfer.

Thus information-theoretic security of a certain requirement against arbitrary active attacks is roughly defined as follows: For *all* strategies of the honest users and *all* strategies of the attacker, the probability that the sequence of inputs and outputs of the correct processors does not fulfil the requirement is exponentially small.

More precisely, this statement is made in the following ensemble of probability spaces: The correct processors have fixed probabilistic programs if we complete their definition by prescribing that each of them engages in at most one transaction at any given time. If a particular strategy of the honest users and a particular attacker are also given, the system according to Figure 1 is a well-defined system of interacting synchronous probabilistic machines. For each initial state, and thus for each tuple of system parameters, it defines a probability space on its runs (traces) for any given number U of time units. In each such probability space, the sequence of in- and outputs of the correct processors is a random variable, and we can define the error probability to be the probability that this sequence does not fulfil the given requirement, i.e., is not an element of a certain set.

We use the following definition of "exponentially small": Let the scheme have a security parameter σ and a tuple *sys_pars* of other system parameters. Then there exists a polynomial *pol* such that for all parameters σ and *sys_pars*, all finite values U, and all user and attacker strategies, the error probability in the resulting probability space is bounded by

$$pol(sys\_pars, U) \cdot 2^{-\sigma}.$$

Such a definition is relatively robust against transformations; in particular, if two requirements are fulfilled in this sense, so is their conjunction.[5] The dependency on U is necessary because even a simple authentication code does not really work if the correct users test exponentially many supposedly authenticated messages. Typically, a concrete polynomial *pol* is known, and one can therefore use $\sigma \geq \sigma^* + \log_2(pol(sys\_pars, U))$ if the actual error probability is to be bounded by $2^{-\sigma^*}$.

Pseudosignatures fulfil the four minimal requirements on authentication systems information-theoretically with an exponentially small error probability in this sense. However, to guarantee finite transferability, the behavior of the honest users must be restricted more strongly, because the error probability would otherwise grow in the order of $2^U \cdot 2^{-\sigma}$. We therefore need an additional parameter *maxtest* in initialization, and any correct recipient's processor will carry out at most *maxtest* tests of pseudosignatures corresponding to this initialization.

3 Construction of Pseudosignatures

In the following, we present information-theoretically secure pseudosignatures. Our scheme is a generalization of the scheme in [CR91], which dealt with the special case of only one transfer and did not consider active attacks on recipients.[6] Furthermore, we improve some details of that scheme.

The pseudosignatures we construct have a fairly simple structure: As with ordinary digital signature schemes, authentications and transfers are noninteractive and transferring means passing a received signature on. Initialization, on the other hand, is highly interactive.

3.1 Overview

The basic idea to achieve unforgeability is that all the recipients' processors must have different test keys, i.e., there is no public key. Thus an unrestricted attacker does not know how a recipient's processor tests a pseudosignature, and therefore the brute-force forging algorithm mentioned in Section 2.3 is no longer applicable. A first idea is that the originator's processor uses an independently chosen key of an information-theoretically secure authentication code with each recipient's processor, and that a pseudosignature consists of n parts, called

---

[5] It can be generalized further, e.g., by allowing a constant $c < 1$ or even a rational transformation in the exponent.

[6] In Section 5 of [CR91], a scheme is sketched that should work for general finite transferability. Implicitly, this scheme assumes that the originator of the message is honest, and thus this extension is not applicable here. (For readers familiar with [CR91, Sect. 5]: The hash functions used there are useful only if they are unknown to the faulty processors, but are known to the originator.)

minisignatures, one for each potential recipient. Each recipient's processor can test this pseudosignature by testing the corresponding minisignature. Forgery is obviously only possible with exponentially small probability. However, this scheme offers no transferability if the originator's processor is faulty: A dishonest originator can issue a pseudosignature in which the minisignature for the first recipient is correct, but those for all other recipients are wrong.

The idea in [CR91] that would guarantee at least one transfer was therefore that the originator should not know which minisignature is for whom and that there are many minisignatures for each recipient. Assume for a moment that this can be done. Then no matter how many wrong and correct minisignatures a cheating originator puts in a pseudosignature, it is highly improbable that all the minisignatures for the first recipient are correct, whereas all those for a certain second recipient are wrong. Hence the processors of first and second recipients need only use different tests:

- The first recipient's processor, i.e., on input (*'test'*, msg, id_S, 0), accepts a pseudosignature if *all* the minisignatures intended for it are correct.
- The second recipient's processor, i.e., on input (*'test'*, msg, id_S, 1), accepts a pseudosignature if at least *one* of the minisignatures intended for it is correct.

Finite transferability for larger λ will be achieved by providing additional different tests.

Now an initialization protocol remains to be shown in which the originator's processor exchanges many keys of an authentication code with the recipients' processors, but the originator cannot find out which of them were exchanged with whom. Essentially, instead of the originator's processor, the recipients' processors generate the keys intended for them and send them to the originator's processor anonymously. The basic method for information-theoretically secure anonymous sending over an untrusted network is the DC protocol in [C88]. It is combined with several measures to ensure that all the recipients' processors succeed in sending a sufficient number of keys even if the faulty processors try to use the anonymous channel all the time, that only the originator's processor learns these keys, and that no processor is falsely eliminated as faulty. Hence, effectiveness and correctness of initialization in the pseudosignature scheme are nontrivial. We therefore start our description of the construction details with the details of anonymous sending.

3.2 Anonymous Channel

An anonymous channel is a distributed protocol in which each processor P_i has an input x_i, and the set of the inputs, $\{x_1, \ldots, x_n\}$, is the output of either all processors or one specific processor. The former is called anonymous broadcast, the latter anonymous many-to-one communication. We assume that a set is represented in lexicographical order. No additional information must become known. As we will need the anonymous channel only in the precomputation phase of the agreement protocol, broadcast and secret channels are available to implement it.

First, let us review the basic DC protocol. It achieves that the sum rather than the set of the inputs is sent without releasing additional information. The output is only correct if all processors act correctly.

Scheme 1. Chaum's DC protocol for the message space $F = \mathrm{GF}(2^u)$.

Phase 1 (Key exchange).

For all pairs (P_i, P_j) of processors with $i < j$:

P_i randomly chooses a *DC key* $K_{i,j} \in F$ and sends it to P_j on a secret channel.

P_j calls the received DC key $K_{j,i}$. If it does not receive one, it sets $K_{j,i} := 0$.

Phase 2 (DC round). Each processor P_i has a local input $x_i \in F$ for this phase.

Each processor P_i computes and broadcasts its *local sum*

$$O_i := x_i + K_{i,1} + \ldots + K_{i,n}.$$

Each processor computes and locally outputs the *global sum*

$$\Sigma := O_1 + \ldots + O_n.$$

By local inputs and outputs of a processor, we mean the values exchanged with its user.

Lemma 1 (Chaum's DC Protocol).

a) If all processors are correct, then

$$\Sigma = x_1 + \ldots + x_n.$$

b) Regardless of how the faulty processors behave, they cannot obtain more information about the local inputs x_i of the correct processors than their sum.

c) If the protocol is modified so that one processor, P_i, does not broadcast its local sum, then P_i can still compute the global sum, but the other processors obtain no information whatsoever about the local inputs. ♦

Proof. Parts a) and b) follow from [C88], and c) (and also a)) can easily be verified. □

Now the protocol is extended such that the set of all inputs, instead of their sum, can be received, and that faults can be detected and localized. We only do this for the case with one specific recipient, as in Lemma 1c). Later, this will be the originator of the pseudosignature; we call the corresponding processor S. Furthermore, we exploit that only random messages will be sent. One advantage of random messages is that they need not be kept secret in the fault localization procedure, another will be seen in Lemma 2b). (The case of normal messages is sketched in [PW92] for an anonymous broadcast channel.) Within the pseudosignature scheme, the random messages will be the authentication keys, and if a fault has been detected and localized, the protocol will be repeated with new keys.

Scheme 2. Anonymous many-to-one channel to processor S with error localization for random messages from a message space $X \subseteq F^* = GF(2^u)\backslash\{0\}$ and with a security parameter γ.

Phase 1 (Key exchange).

Phase 1 of Scheme 1 is used $2n$ times in parallel to exchange keys for $2n$ DC rounds.

Phase 2 (Sending messages). Each processor P_i has a local input x_i randomly chosen from X for this phase.

[1] Phase 2 of Scheme 1 is executed $2n$ times in parallel, except that S does not output its local sums. For the h-th execution, processor P_i uses

$$x_i^{2h-1}$$

as its local input, and S obtains the global sum Σ_{2h-1}.

[2] S uses the first n global sums, $\Sigma_1, \Sigma_3, \ldots, \Sigma_{2n-1}$, to compute a set $Y = \{y_1, \ldots, y_l\} \subseteq F^*$ with $l \leq n$ such that

$$\Sigma_{2h-1} = y_1^{2h-1} + \ldots + y_l^{2h-1}$$

for $h = 1, \ldots, n$. Any such set is called a *solution*. The algorithm is described in [BB90, B71, R80]; it has a security parameter γ and an error probability of at most $2^{-\gamma}$.

If this worked and if $\Sigma_{2h-1} = y_1^{2h-1} + \ldots + y_l^{2h-1}$ also holds for $h = n+1, \ldots, 2n$, S locally outputs Y as the set of received messages. Otherwise it broadcasts a *complaint message*.

[3] If S has broadcast a complaint message, each processor P_i broadcasts its DC keys and local input used in Phases 1 and 2. Every processor P_j can now verify the entire behavior of the other processors so far. If P_j finds any fault in the behavior of P_i, it locally outputs "i faulty". If two processors P_i, $P_{i'}$ do not agree on their common DC key $K_{i,i'}$, then P_j outputs "(i, i') faulty". If P_j finds neither a faulty processor nor a faulty key, it outputs "S faulty".

Power sums were first used to reconstruct individual messages in [BB90], but only the first n of them were used and thus faults could not be detected. A different mechanism to detect faults is used in [CR91], but its security is not proven and it is not more efficient than ours in the given application. Once a fault has been detected, the localization of a faulty processor or key is as in [C88].

Lemma 2 (Anonymous channel with error localization).

a) If all processors are correct and their local inputs are elements of the correct set, Scheme 2 is completed without complaint with probability at least $1-2^{-\gamma}$, and the set Y is the set of local inputs x_i that occur an odd number of times.

b) If S is correct, it either complains or its set Y contains all the local inputs of the correct processors that occur an odd number of times, with probability at least $1 - 2n^2|X|^{-1}$. We call the opposite a *successful disruption.*

c) If S is correct, the attacker obtains at most one bit of information about the local inputs of the correct processors.

d) After a complaint, each correct processor identifies at least one processor or key as being faulty.

e) The probability that a correct processor P_i is identified as being faulty by another correct processor is at most $2^{-\gamma}$. Actually, this can only happen to S. Keys between two correct processors are never identified as being faulty by other correct processors.

f) Even if S is faulty, the attacker cannot obtain more information about the local inputs x_i of the correct processors than the multiset of them, except if there is a complaint. ♦

Proof. Without loss of generality, let the correct processors be $P_1, \ldots, P_k$.

a) It follows from [BB90, B71, R80] that for any values $\Sigma_1, \Sigma_3, \ldots, \Sigma_{2n-1}$, the set of equations described in Step [2] has at most one solution. If a solution exists, the algorithm described there either finds it or, with probability at most $2^{-\gamma}$, stops with an error message. If all processors are correct, the local inputs are a solution, and thus Part a) follows.

b) The faulty processors can only disrupt by outputting wrong local sums. Altogether, this means that they select values $\Delta_1, \Delta_3, \ldots$ such that the global sums are

$$x_1{}^{2h-1} + \ldots + x_k{}^{2h-1} + \Delta_{2h-1}$$

for $h = 1, \ldots, 2n$. If S does not complain, pairwise distinct values $y_1, \ldots, y_l \in F^*$ with $l \leq n$ exist such that

$$x_1{}^{2h-1} + \ldots + x_k{}^{2h-1} + \Delta_{2h-1} = y_1{}^{2h-1} + \ldots + y_l{}^{2h-1}$$

for $h = 1, \ldots, 2n$. In GF(2^u), this is equivalent to

$$\Delta_{2h-1} = x_1{}^{2h-1} + \ldots + x_k{}^{2h-1} + y_1{}^{2h-1} + \ldots + y_l{}^{2h-1}. \qquad (1)$$

The disruption is successful if at least one value x_{i^*} chosen by an odd number of correct processors is not among $y_1, \ldots, y_l$. This implies that x_{i^*} occurs in (1) an odd number of times.

By the proof of Part a), there is at most one solution to the power sum equations defined by $(\Delta_1, \Delta_3, \ldots, \Delta_{4n-1})$, i.e., at most one set $Z = \{z_1, \ldots, z_{l^*}\} \subseteq F^*$ with $l^* \leq 2n$ such that

$$\Delta_{2h-1} = z_1{}^{2h-1} + \ldots + z_{l^*}{}^{2h-1}$$

for $h = 1, \ldots, 2n$. By (1), we have a solution in $x_1, \ldots, x_k, y_1, \ldots, y_l$ if values occurring an even number of times are deleted. Hence they must be the set Z, and in particular, x_{i^*} is in Z. By Lemma c), the faulty processors have no information about $x_1, \ldots, x_k$, even if they try to base their own outputs on those of the correct processors by choosing their local sums last. Their values Δ_{2h-1}, and thus $z_1, \ldots, z_{l^*}$, are therefore independent of $x_1, \ldots, x_k$. (In principle, we have now proved that the faulty processors have to guess at least one input x_{i^*} of a correct processor in order to disrupt successfully.)

The probability of $x_{i^*} \in Z$ for fixed i^* is at most $2n|X|^{-1}$. Thus the overall probability of a successful disruption is at most $2n^2|X|^{-1}$.

c) It is clear by Lemma 1c) that the attacker obtains no information before the possible complaint message, which is only one bit long. (Actually, it contains only an exponentially small amount of information, because with any attacker strategy one output is very unlikely.)

d) Obvious: If no other fault is found, S will be blamed.

e) The only possibility is that S is identified as being faulty because of a wrong complaint. If no other processor or key is identified as being faulty in Step [3], the global sums that S used in

Step [2] have a solution in the local inputs the processors later claim to have had. Thus S would only have complained if the algorithm used to solve the power-sum equations failed although a solution exists. Hence, by the proof of Part a), the probability is at most $2^{-\gamma}$.

f) By Lemma 1, the faulty processors do not obtain any information about $x_1, \ldots, x_k$ except for the power sums $\Sigma^*_{2h-1} = x_1^{2h-1} + \ldots + x_k^{2h-1}$ for $h = 1, \ldots, 2n$. As each permutation of $(x_1, \ldots, x_k)$ yields the same power sums, the faulty processors may find out the values x_i, but they obtain no information about which x_i comes from which correct processor, i.e., they only have the multiset. (Actually, they have slightly less information, because local inputs chosen by an even number of processors cancel out in all power sums, and thus the faulty processors do not learn them.) □

3.3 The Pseudosignature Scheme

As sketched in Section 3.1, we use Scheme 2 to let the processors of future recipients of pseudosignatures send keys of an authentication code anonymously to the future originator's processor.

We only need a one-time authentication code, i.e., $N = 1$. Without loss of generality, the authentication keys are coded as elements of a set $X \subseteq \mathrm{GF}(2^u)\backslash\{0\}$, where u can be computed as a function $keylen(l, \tau)$ of the message length and the security parameter. Furthermore, we assume that membership in X can be decided in time polynomial in l and τ, and that gen generates keys with uniform distribution in X. This implies $|X| \geq 2^\tau$. In particular, we can use the authentication code of [WC81] and as its underlying strongly universal$_2$ class of functions the multiplicative scheme of [CW79] over $\mathrm{GF}(2^{\tau+1})$. Then the length of authenticators is $\tau+1$ bits, and $keylen(l, \tau)$ is logarithmic in l and linear in τ.

We only construct a one-time pseudosignature scheme, i.e., $N = 1$, as this will be sufficient for Byzantine agreement. Some improvement over mere repetition is conceivable for larger N, but even with simple authentication codes, it is easy to see that the complexity of initialization grows at least linearly in N.

Scheme 3. One-time pseudosignature scheme with finite transferability for a message space $\{0, 1\}^l$ with security parameter ρ.

Let an authentication scheme be given as above. It will be used for the given message space $\{0, 1\}^l$.

Initialization for parameters λ, the number of possible transfers, and *maxtest*, the maximum number of tests a recipient's processor will carry out.

Let

$$\rho^* := \rho + 2\mathit{maxtest},$$

and instantiate the authentication scheme with the security parameter

$$\tau := \rho^* + \log_2(\rho^*) + 2\log_2(\lambda+1).$$

Furthermore, let $u := keylen(l, \tau)$ and $F = \mathrm{GF}(2^u)$. Define the acceptance difference Δ and the multiplicity m as

$$\Delta := \lceil 2 \ln(2)\, (\lambda+1)\rho^* \rceil,$$
$$m := \lambda\Delta + 1.$$

Each processor P_i executes in parallel for $k := 1$ to m:

It repeats the following three steps until the originator's processor, S, does not complain or until P_i regards S as having been eliminated (see Step [3]):

[1] If $P_i \neq S$, it randomly chooses an authentication key $ak_{k,i}$, coded as an element of F^*.

[2] It takes part in Scheme 2 with all processors, message space F^*, and security parameter $\gamma := \tau$ to send this key anonymously to S.

[3] If S has complained and a processor or key is identified as being faulty, they are eliminated forever. (Eliminating $K_{j,j'}$ means that P_j and $P_{j'}$ will never use a common DC key again.) If no keys are left between two groups of processors, P_j regards the processors that are not in its group as eliminated.

If S has not complained, it stores the set of received values that are correct authentication keys, i.e., members of X, say

$$Y_k = \{y_{k,1}, \ldots, y_{k,l}\},$$

and each of the other processors P_i stores its key $ak_{k,i}$ from this iteration.

P_i decides that initialization was successful, i.e., $acc = true$, if it does not regard S as eliminated; P_i's output $ids_{R,out}$ is the set of recipients that it does not regard as eliminated.

Authentication.

Originator. Given a message *msg* from the message space as a local input, S computes the pseudosignature on *msg* as

$$\psi := (\psi_1, \ldots, \psi_m),$$

where each so-called *section* ψ_k is

$$\psi_k := \{\, auth(ak, msg) \mid ak \in Y_k \}.$$

Recipient. We immediately show tests for any λ^*: If P_i is given a message *msg*, a supposed pseudosignature $\psi = (\psi_1, \ldots, \psi_m)$ on it, and $\lambda^* \in \{0, \ldots, \lambda\}$, it first verifies that ψ is of the correct format and then determines the acceptance number,

$$acc\_no := \left| \{k \in \{1, \ldots, m\} \mid auth(ak_{k,i}, msg) \in \psi_k\} \right|$$

i.e., the number of correct minisignatures for P_i in ψ. It λ^*-accepts if and only if

$$acc\_no > (\lambda - \lambda^*)\Delta.$$

Transfer.

A processor that has received a pseudosignature ψ on a message *msg* can simply forward it; the recipient tests it as shown above.

Note that accepting for $\lambda^* = 0$, i.e., directly from the originator, means $acc\_no = m$, i.e., all minisignatures for this recipient must be correct.

Lemma 3 (Termination and round complexity). All the algorithms in Scheme 3 are polynomial-time. In particular, the initialization deterministically terminates in fewer than $2n^2$ rounds. In the correct case, i.e., if no processor is faulty, only three rounds are needed. Scheme 2 is executed at most $mn^2/2$ times.

If w pseudosignatures are initialized sequentially and if the elimination of a DC key or a faulty processor is valid for all initializations, all initializations together need fewer than $3w+2n^2$ rounds. ♦

Proof. After every complaint, either a key or a processor that still has at least one key is eliminated. Hence, after fewer than $n(n-1)/2$ complaints, no keys between correct and faulty processors are left and the correct processors can never be disturbed again. Hence there are fewer than $n(n-1)/2+w$ sequential repetitions of Steps [1] to [3]. This implies fewer than $mn^2/2$ executions of Scheme 2 in one initialization. As to rounds, only Step [2] needs any, either three or four: key exchange, basic DC protocol, complaint, and broadcasting keys and local inputs if there was a complaint. (One could reduce this further by performing the entire key exchange in one initial round.) □

Theorem 1. Scheme 3 is a secure pseudosignature scheme. We summarize the five security statements here with their precise maximum error probabilities for any attackers and users. (We also give them in terms of the inner security parameter ρ^* to prepare for a more efficient choice of ρ^* within the Byzantine agreement protocol.)

a) Effectiveness of initialization. If the processors of the originator and at least one recipient are correct, the correct processors finish initialization successfully with probability at least $1 - n^2 2^{-\rho^*} \geq 1 - n^2 2^{-\rho}$.

b) Correctness of initialization. All correct processors always make the same output, and the identities of all correct recipients' processors are in $ids_{R,out}$.

c) Effectiveness of authentication. If S is correct, the probability that initialization is successful and a correct processor will nevertheless not accept a pseudosignature is at most $3n^4 2^{-\rho^*} \leq 3n^4 2^{-\rho}$.

d) Unforgeability. The probability that a correct recipient's processor accepts a message for any $\lambda^* \leq \lambda$ that the correct originator has not yet authenticated is at most $4n^2 maxtest \cdot 2^{-\rho^*} \leq 4n^2 2^{-\rho}$ (for all recipients together).

e) Finite transferability. Let P and Q be two correct recipient's processors. The probability that one of them λ^*-accepts a pseudosignature for any $\lambda^* < \lambda$ that the other would not even (λ^*+1)-accept is at most $2^{2maxtest} 2^{-\rho^*} = 2^{-\rho}$. ♦

Proof of Parts a)-d). For all parts, note that $m \leq 2(\lambda+1)^2 \rho^*$ and thus $m 2^{-\tau} \leq 2 \cdot 2^{-\rho^*}$.

a) The contrary means that a correct processor eliminates S or their common key. By Lemma 2e), this happens with a probability of at most $2^{-\tau}$ in each execution of Scheme 2, and thus, with Lemma 3, with probability at most $1/2mn^2 2^{-\tau} \leq n^2 2^{-\rho^*}$ altogether.

b) The former is clear because all these decisions are based on broadcast messages. The latter follows from Lemma 2e).

c) As the authentication code used guarantees error-free effectiveness of authentication, there are only two possibilities why a pseudosignature from S might not be accepted by all other correct processors: Either S has not received all the authentication keys from those processors owing to a successful disruption, or two of these processors chose the same key so that it canceled out. Both possibilities only depend on initialization, and therefore we can omit active attacks, such as letting recipients test other messages. By Lemma 2b), the probability of a successful disruption is at most $2n^2|X|^{-1}$ in each execution of Scheme 2, where $|X| \geq 2^\tau$, and thus at most $mn^4 2^{-\tau}$ altogether. The probability that two processors choose the same key is at most $n^2|X|^{-1}$ in any single execution of Step [1], and thus at most $(mn^4/2)2^{-\tau}$ altogether. The sum of the two probabilities is at most $3n^4 2^{-\rho^*}$.

d) (Unforgeability.) The worst case of attack is when the attackers first let the originator authenticate another message, msg^*, to obtain one pseudosignature ψ^*, and then let the recipients test as many forged signatures as possible, hoping that at least one will be accepted. P_i only accepts a pair (msg, ψ) if at least one minisignature in it is the correct value $auth(ak_{k,i}, msg)$.

As ψ^* only contains $auth(ak_{k,i}, msg^*)$, the probability that the attackers guess any particular $auth(ak_{k,i}, msg)$ correctly would be at most $2^{-\tau}$ by the security of the authentication code, if the attackers learned nothing about $ak_{k,i}$ during initialization. By Lemma 2c), they learn at most one bit. This can at most increase the success probability by a factor of 2, because they could also guess the bit.

Each of the n recipients tests at most $maxtest$ pseudosignatures with mn minisignatures each. Thus the overall success probability of the attackers is at most $n \cdot maxtest \cdot mn 2^{-\tau+1} \leq 4n^2 maxtest \cdot 2^{-\rho^*}$. □

For transferability, we first prove the following lemma:

Lemma 4 (Finite transferability under passive attacks). Let P and Q be two correct recipient's processors and $\lambda^* < \lambda$. If one initialization is carried out and then the attackers immediately construct a pair (msg, ψ), the probability that P λ^*-accepts it, while Q does not even (λ^*+1)-accept it, is at most $2^{-\rho^*}$. ♦

Proof. We only consider the worst case, where P and Q are the only correct processors. In particular, S is faulty, and the faulty processors therefore know the correct pseudosignature ψ^* on msg. Furthermore, for each section of the pseudosignature, they know which two authentication keys belong to P and Q. However, by Lemma 2f), they have no information about which key was chosen by P and which by Q. For each of the m sections, the faulty processors can decide to change none, one, or two of the two correct minisignatures, i.e., to replace them by incorrect ones. For reasons of symmetry, we can describe this decision by just two parameters:

g denotes the number of sections in which both minisignatures are changed, and

h denotes the number of sections in which exactly one minisignature is changed.

With probability $\binom{h}{y} 2^{-h}$, within the h sections in which one minisignature was changed, exactly y changed minisignatures belong to P. Let B_h denote the binomial distribution with parameter h and probability 1/2, i.e.,

$$B_h(k) = \sum_{y \leq k} \binom{h}{y} 2^{-h}.$$

Let $x := \lambda^* \cdot \Delta$. The faulty processors are successful if they have changed at most x minisignatures for P and more than $x+\Delta$ minisignatures for Q, i.e., $g+y \leq x$ and $g+(h-y) > x+\Delta$, which means $y \leq \min\{x-g, g+h-x-\Delta-1\}$. This happens with probability

$$\mathrm{P}(g, h) := B_h(\min\{x-g, g+h-x-\Delta-1\}) \leq B_h\left(\frac{(x-g) + (g+h-x-\Delta-1)}{2}\right) = B_h\left(\frac{h}{2} - \frac{\Delta+1}{2}\right).$$

For each $\delta \geq 0$, it follows from [ES74, Equation (3.8)] that

$$B_h\left(\frac{h}{2} - \delta - \frac{1}{2}\right) \leq \sum_{z<h/2-\delta} \binom{h}{z} 2^{-h} < \exp\left(-\frac{2\delta^2}{h}\right).$$

This implies

$$\mathrm{P}(g, h) < \exp\left(-\frac{\Delta^2}{2h}\right).$$

To prove $\mathrm{P}(g, h) \leq 2^{-\rho^*}$, it therefore remains to be shown that $\Delta^2 \geq 2h\rho^* \ln(2)$. As $m \geq h$, it is sufficient if $\Delta^2 \geq 2m\rho^* \ln(2) = 2(\lambda\Delta+1)\rho^* \ln(2)$, which is clearly satisfied as $\Delta \geq 2(\lambda+1)\rho^* \ln(2)$. □

To see why active attacks on recipients make a big difference for transferability, consider the following example: Let P and Q again be the correct processors and ψ^* a correct pseudosignature. In each round, the faulty processors change exactly one minisignature in ψ^* and ask P whether it 0-accepts the result ψ. The answer is *true* iff the changed minisignature did not belong to P. Therefore, after m interactions, the faulty processors know exactly which minisignatures belong to P. Hence, they can construct another ψ that P will 0-accept, but Q will not accept at all. This is why *maxtest* had to be restricted.

Proof of Theorem 1e). Once more, we consider the worst case, where only P and Q are correct. Furthermore, we consider the combined strategy of the attacker and the honest users, i.e., in Figure 1, everything except the two correct processors is considered as one big box.

The strategy after initialization can be described by a tree of the following type: Each node is labeled by a quadruple $(P^*, msg, \psi, \lambda^*)$, where $P^* \in \{P, Q\}$ and $\lambda^* \leq \lambda$. This means that P^* is given the local input $(\text{'}test\text{'}, msg, S, \lambda^*)$ and is sent the pseudosignature ψ. Each node has two sets of children, corresponding to the next (probabilistic) step of the strategy after the reaction $acc = true$ or $false$ of P^*, respectively. The tree has at most $2maxtest$ levels, because P and Q test at most *maxtest* pseudosignatures each.

We can make the strategy tree deterministic without decreasing its probability of success: Each probabilistic decision is replaced by deterministically taking the choice with the maximum probability of success. The resulting tree is binary and still has at most $2maxtest$ levels and thus fewer than $2^{2maxtest}$ nodes, i.e., pseudosignatures.

The success probability of the active attack is now bounded above by assuming that after any initialization the attackers could try all the pseudosignatures of the corresponding tree on P and Q. With Lemma 4, this yields an error probability of at most $2^{2maxtest} 2^{-\rho^*}$. (Note that the only remaining probabilistic choices are those made by P and Q during initialization. Thus in each event, i.e., for any fixed set of these choices, it will be determined before the active attack whether a certain pseudosignature is λ^*-acceptable to P or Q; the whole information from the active attack lies in the decisions whether this node is reached in this event.) □

4 Byzantine Agreement

We modify the authenticated Byzantine agreement protocol from [DS83, Theorem 3] so that it can be implemented with pseudosignatures rather than with ordinary digital signatures. In particular, we determine the levels of acceptance and minimize the information that can be gained by active attacks. Actually, we will show that active attacks on the pseudosignatures within this Byzantine agreement protocol have a much smaller influence than in general, so that the basic security parameters only have to grow logarithmically in n (which also determines the number of rounds), and not linearly, as one would expect from Section 2.4.2 and Scheme 3.

Scheme 4. Byzantine agreement for n processors, where $t < n$ may be faulty,
with the message space $\{0, 1\}^l$ and a security parameter σ
and the transmitter P_1.

Precomputation phase.

The initialization of Scheme 3 is executed $2n$–1 times in parallel for $\lambda := t$, message space $\{0, 1\}^l$, *maxtest* $= 2n$, and inner security parameter

$$\rho^* := \sigma + \lceil 18 \log_2(n) \rceil + 6.$$

Each processor plays the role of the originator's twice, and the identities used are (i, A) and (i, B) for $i := 1, \ldots, n$, except that only one initialization with $(1, A)$ is needed for the transmitter. The pseudosignatures corresponding to these initializations are called A- and B-pseudosignatures, respectively.

Main phase. The transmitter, P_1, now has a value $v \in \{0, 1\}^l$ as its local input.

[0] P_1 computes its A-pseudosignature ψ on v and forwards $(v, \{(1, A, \psi)\})$ to all the other processors.

Each $P_i \neq P_1$ initializes two sets ACC_i and OLD_i as empty. ACC_i is called the set of accepted values, OLD_i the set of old relay attempts.

For $k = 1, \ldots, t{+}1$ and each processor $P_i \neq P_1$:

[k] Each processor $P_i \neq P_1$ evaluates the messages that have been sent to it in round [k–1]:

a) It accepts at most one message M from P_1 in round [1] and from each processor $P_j \neq P_1$ in the other rounds. It verifies that M is *well-formed*, which means that

$$M = \left(v, \{(i_1, \alpha_1, \psi_1), \ldots, (i_k, \alpha_k, \psi_k)\}\right),$$

where v is a value from the message space, the values $i_{j'}$ are k different processor numbers not including i, all $\alpha_{j'} \in \{A, B\}$, and the pair $(1, A)$ occurs among the pairs $(i_{j'}, \alpha_{j'})$. Intuitively, this means that P_j claims that v has been signed by the transmitter and k–1 other processors.

b) P_i decides for each well-formed message M whether it is *new*, i.e., contains a triple (j, α, ψ) where P_j is the sender of M and $(j, \alpha) \notin OLD_i$. If yes, P_i adds (j, α) to OLD_i.

c) P_i decides which of the new messages M are *k-consistent*. This means that P_i additionally $(k–1)$-accepts each value $\psi_{j'}$ as a pseudosignature on v for the identity $(i_{j'}, \alpha_{j'})$[7]. Let $Cons_{i,k}$ be the set of these messages, and let it be lexicographically ordered. P_i adds all the values v that are contained in a message in $Cons_{i,k}$ to ACC_i.

d) If $k \leq t$ and $Cons_{i,k} \neq \emptyset$, P_i possibly *relays* messages. Relaying a message M of the form described above means that P_i constructs $M^* = \big(v, \{(i_1, \alpha_1, \psi_1), \ldots, (i_k, \alpha_k, \psi_k), (i, \alpha, \psi)\}\big)$, where $\alpha = A$ if this is the first message that P_i relays and $\alpha = B$ otherwise and ψ is P_i's α-pseudosignature on v, and forwards it to those processors whose number j was not contained in M.

1. If P_i has not relayed any message before, it relays the first two messages in $Cons_{i,k}$ that contain different values, or just the first one, if all messages contain the same value.
2. If P_i has relayed exactly one message, which contained a value v', then it relays the first message in $Cons_{i,k}$ that contains a value $v'' \neq v'$, if there is one.

The local output of the transmitter P_1 is its own local input v. Any other processor P_i outputs a value v iff $ACC_i = \{v\}$, and otherwise, "transmitter faulty".

The use of the set OLD_i guarantees that P_i reacts on at most two messages from each of the other processors. Thus the choice of $maxtest = 2n$ is sufficiently large, because each well-formed message contains at most one pseudosignature corresponding to a particular instantiation.

The correctness of Scheme 4 is proved in two steps: First we show that it is always correct if no error occurs with the pseudosignatures. Secondly, we show that the probability that no such error occurs is at least $1–2^{-\sigma}$, and thus the error probability of the Byzantine agreement protocol is at most $2^{-\sigma}$.

Lemma 5 (Deterministic version of Theorem 2). In any execution of Scheme 4 in which all five requirements are fulfilled for all $2n–1$ instantiations of the pseudosignature scheme, correct agreement is achieved. ♦

Note that here the requirements in their original form, i.e., as predicates on sequences of inputs and outputs, are meant, not their embedding into probabilistic statements as in Section 2.4.2.

Proof. The proof follows the outline in [DS83, Theorem 3].

a) Correctness. If the transmitter P_1 is correct, it sends its value as prescribed in Round [0]. This message is 1-consistent for all correct processors P_i, and thus they all add v to ACC_i in Round [1]. As P_1 does not send further messages, unforgeability implies that no k-consistent message with a value $v' \neq v$ occurs. Thus all correct processors output v.

b) Consistency. We can now assume that P_1 is faulty.

1. First we show that if a correct processor P_i relays a message M containing a value v in any round $[k]$ (which implies $k \leq t$), then $v \in ACC_j$ for all correct processors P_j at the end.

---

[7] In [DS83], the signatures in k-consistent messages are not just on v, but on the entire message. Our construction results in smaller messages to be signed and does not restrict the fault tolerance.

If j already occurred in M, then, by k-consistency, P_i has accepted a pseudosignature by P_j on v. By unforgeability, P_j made this pseudosignature, and it could only have done so in relaying a message M' that contained v. In that round, P_j also added v to ACC_j.

If j did not occur in M, then P_i forwards M^* to P_j. As M was well-formed for P_i in Round $[k]$, M^* is well-formed for P_j in Round $[k+1]$. M^* is also new for P_j, because P_i only makes one A- and one B-pseudosignature. Finally, it is $(k+1)$-consistent: By finite transferability, P_j k-accepts all the pseudosignatures that were already contained in M, and by effectiveness of authentication, it would even 1-accept the new pseudosignature.

2. Now we prove the following statement: If $v \in ACC_i$ for a correct processor P_i, then $v \in ACC_j$ or $|ACC_j| \geq 2$ for all correct processors P_j at the end.

Let $[k]$ be the round in which v is added to ACC_i, and let M be a message in $Cons_{i,k}$ that contains v. If P_i relays M, then the statement follows from 1. If P_i does not relay M, this may be for three reasons:

- $k = t+1$. Then M contains $t+1$ processor numbers and thus the number of at least one correct processor P_h. This implies that P_h has relayed a message containing the value v in a previous round, and 1. implies $v \in ACC_j$ for all correct processors.
- P_i has already relayed another message containing v, or does so now. Again, 1. yields $v \in ACC_j$ for all correct processors.
- P_i has already relayed messages with two different values. Then, by 1., all other correct processors have these two values in ACC_j.

3. Now we can show consistency: If $|ACC_i| \geq 2$ for any correct processor, then 2. immediately implies that $|ACC_j| \geq 2$ for all correct processors, and they all decide "transmitter faulty". Hence, if $ACC_i = \{v\}$ for one correct processor, the only remaining possibility for the other correct processors is $ACC_j = \{v\}$, as well. Otherwise, $ACC_j = \emptyset$ for all of them, and they decide "transmitter faulty". □

Theorem 2 (Byzantine agreement). For any $t < n$, Scheme 4 achieves Byzantine agreement with an error probability of at most $2^{-\sigma}$. ♦

Proof. Given Lemma 5, the error probability of the Byzantine agreement protocol is at most the joint error probability of the $2n-1$ instantiations of the signature scheme. As these instantiations are independent, we can compute the joint error probability of one of them and multiply by $2n$.

a) For the first four requirements, Theorem 1 immediately yields a joint error probability of at most

$$(1 + 3n^2 + 4maxtest)\, n^2\, 2^{-\rho^*} \;=\; (1 + 3n^2 + 4n)\, n^2\, 2^{-\rho^*}.$$

b) For finite transferability, we first consider two particular correct processors P and Q. We consider an overly pessimistic case in which P and Q are the only correct processors and the attackers even know P's and Q's secrets from the initialization of all other pseudosignatures. Again, we count pseudosignatures in a strategy tree. However, we can now exploit that in the Byzantine agreement protocol, P and Q do not tell the attackers whether they accepted an individual pseudosignature. Instead, their only reaction is to relay up to two messages each. This allows us to construct a different strategy tree, *Tree*, and to prove much smaller error probabilities.

Each node of *Tree* represents one round and is labeled by the new well-formed messages that P and Q have received in this round. Other messages need not be considered, because the attackers can evaluate on their own that those messages are not new and well-formed and will be discarded by P and Q.

The children of a node of Round $[k]$ with $k \leq t$ correspond to the possible reactions by P and Q. Each one can be described unambiguously by specifying which messages P and Q relay. (The pseudosignature that P or Q adds before relaying is already known in this pessimistic case.) We call the child corresponding to the case that neither P nor Q relays a message the leftmost child.

We now count the paths in this tree. Altogether, P and Q relay at most two messages each. Thus on each path, there are at most 4 branches to the right, i.e., not to the leftmost child. In each round, P and Q accept at most two messages from each other processor, thus together at most $2n-2$ each. At the beginning, P and Q can relay up to two messages each. These are

$$\binom{2n-2}{0} + \binom{2n-2}{1} + \binom{2n-2}{2} \leq 2n^2$$

different possibilities for P and Q each. Therefore, until the first branch to the right, each node has at most $c_0 := 4n^4$ children. Each branch to the right reduces the number of possible reactions. It is easy to see that after the first branch to the right, there are at most $c_1 := 4n^3$ possibilities left, after the second, at most $c_2 := 4n^2$, after the third, at most $c_3 := 2n$, and after the fourth, $c_4 = 1$. Each path can be uniquely described by specifying in which of the $n-1$ possible depths the $r \leq 4$ branches to the right occur, and for the i-th branch, which of the at most $c_{i-1}-1$ branches to the right is chosen. Thus, for $r = 0, 1, 2, 3, 4$, there are

$$paths_r \leq \binom{n}{r} c_0 \ldots c_{r-1}$$

paths with exactly r branches to the right. After some computation, this yields $paths \leq 9n^{14}$ for the overall number of paths in *Tree*.

Each path in *Tree* contains at most $2maxtest = 4n$ pseudosignatures. Thus, there are at most $36n^{15}$ pseudosignatures in *Tree*. As in the proof of Theorem 1e), we can bound the success probability of the active attack by the probability that any pseudosignature in *Tree* violates finite transferability, and thus by $36n^{15}2^{-\rho^*}$.

As there at most $n^2/2$ sets of two correct processors $\{P, Q\}$, the joint error probability of finite transferability for all correct processors with respect to one instantiation of the pseudosignature scheme is at most

$$P_2 := 18n^{17}2^{-\rho^*}.$$

c) The overall error probability for all instantiations of the pseudosignature scheme is therefore at most (using $n \geq 2$)

$$2n(P_1 + P_2) \leq 2n((1 + 3n^2 + 4n)n^2 2^{-\rho^*} + 18n^{17}2^{-\rho^*}) \leq 37n^{18}2^{-\rho^*} \leq 2^{-\sigma}. \qquad \square$$

Lemma 6 (Complexity). Scheme 4 is polynomial-time in n, l, and σ. The precomputation phase needs 3 rounds in the correct case and $2n^2$ rounds in the worst case. The main phase always needs $t+1$ rounds, and thus at most n. ♦

Proof. Straightforward counting. □

5 Fixed-Length Precomputation

Scheme 4 needs $2n–1$ pseudosignatures for each agreement. Thus the length of its precomputation phase, which uses a physically reliable broadcast network, is proportional to the intended number of executions of the main phase. We now sketch a bootstrapping mechanism so that an (almost) arbitrary number N^* of agreements can be carried out after a precomputation phase of fixed length.

More precisely, the precomputation consists of $O(n^2)$ parallel executions of the precomputation phase of Scheme 4. Secret channels are needed in the main phase now because they cannot be bootstrapped. The error probability of all agreements together is $pol(N^*)2^{-\sigma}$ for some polynomial, which corresponds to the definition of an exponentially small error probability in Section 2.4.2.

Bootstrapping works as follows: In the precomputation phase with a broadcast channel, we perform initialization for a certain number of pseudosignatures. Before we run out of them, we use the Byzantine agreement protocol as the broadcast channel in the initialization for more pseudosignatures. This does not work trivially, because so far, we need $2n–1$ pseudosignatures for one Byzantine agreement, and many Byzantine agreements to initialize one new pseudosignature. However, one can perform any number of pseudosignature initializations in parallel with a fixed number x of broadcasts: All the corresponding messages of these executions are sent in one long message v. The length of v only has a logarithmic influence on the initialization of the pseudosignatures needed to sign it during the Byzantine agreement and therefore poses no problem. Hence, each time one initializes a sufficient number of pseudosignatures for $x+y$ broadcasts, which provides y broadcasts for normal use.

6 Summary

In Section 2, we presented a framework for the definition and different types of authentication systems and used it to define a new type, the so-called pseudosignatures. Even though care with active attacks on recipients must be taken, pseudosignatures can be an information-theoretically secure substitute for ordinary digital signatures in a large class of deterministically polynomial protocols. In Section 3, we showed how pseudosignatures can be constructed.

In Sections 4 and 5, we presented a modification of a well-known authenticated Byzantine agreement protocol and proved that it can be implemented securely with pseudosignatures. This results in the first Byzantine agreement protocol tolerating any number $t < n$ of faulty processors without relying on unproven computational assumptions.

We consider our results constructive proofs of existence rather than practical constructions. An interesting, open question is whether there are more efficient constructions, in particular for pseudosignatures.

Acknowledgments. We are pleased to thank *Birgit Baum-Waidner*, *Manfred Böttger*, *David Chaum*, *Klaus Echtle*, *Maarten van der Ham*, *Leonid Levin*, *Andreas Pfitzmann*, *Rüdiger Reischuk*, and *Sandra Roijakkers* for helpful comments and discussions.

References

[B71] Herbert O. Burton: Inversionless Decoding of Binary BCH Codes; *IEEE Transactions on Information Theory 17/4* (1971) pp. 464-466.

[BB90] Jurjen Bos, Bert den Boer: Detection of Disrupters in the DC Protocol; *Proc. Eurocrypt '89*, Lecture Notes in Computer Science 434, Springer-Verlag, Berlin, 1990, pp. 320-327.

[BMD93] Michael Barborak, Miroslaw Malek, Anton Dahbura: The Consensus Problem in Fault-Tolerant Computing; *ACM Computing Surveys 25/2* (1993) pp. 171-220.

[BPW91] Birgit Baum-Waidner, Birgit Pfitzmann, Michael Waidner: Unconditional Byzantine Agreement with Good Majority; *Proc. 8th Symposium on Theoretical Aspects of Computer Science (STACS)*, Lecture Notes in Computer Science 480, Springer-Verlag, Heidelberg, Germany, 1991, pp. 285-295.

[BS88] Ernest F. Brickell, Doug R. Stinson: Authentication Codes with Multiple Arbiters; *Proc. Eurocrypt '88*, Lecture Notes in Computer Science 330, Springer-Verlag, Berlin, Germany, 1988, pp. 51-55.

[C88] David Chaum: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability; *Journal of Cryptology 1/1* (1988) pp. 65-75.

[CD89] Benny Chor, Cynthia Dwork: Randomization in Byzantine Agreement; *Advances in Computing Research Vol. 5*, JAI Press, Greenwich, Connecticut, 1989, pp. 443-497.

[CR91] David Chaum, Sandra Roijakkers: Unconditionally Secure Digital Signatures; *Proc. Crypto '90*, Lecture Notes in Computer Science 537, Springer-Verlag, Berlin, Germany, 1991, pp. 206-214.

[CW79] J. Lawrence Carter, Mark N. Wegman: Universal Classes of Hash Functions; *Journal of Computer and System Sciences 18* (1979) pp. 143-154.

[DD91] Danny Dolev, Cynthia Dwork: *On-The-Fly Generation of Names and Communication Primitives*; IBM Research Division, Almaden Research Center, San Jose, received Dec. 1991; result already referred to by P. Feldman and S. Micali, STOC 88.

[DH76] Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography; *IEEE Transactions on Information Theory 22/6* (1976) pp. 644-654.

[DS83] Danny Dolev, H. Raymond Strong: Authenticated Algorithms for Byzantine Agreement; *SIAM Journal on Computing 12/4* (1983) pp. 656-666.

[DY90] Yvo Desmedt, Moti Yung: *Asymmetric and Securely-Arbitrated Unconditional Authentication Systems*; submitted to IEEE Transactions on Information Theory, accepted pending revisions, 1990.

[DY91] Yvo Desmedt, Moti Yung: Arbitrated Unconditionally Secure Authentication can be Unconditionally Protected Against Arbiter's Attacks; *Proc. Crypto '90*, Lecture Notes in Computer Science 537, Springer-Verlag, Berlin, Germany, 1991, pp. 177-188.

[ES74] Paul Erdös, Joel Spencer: *Probabilistic Methods in Combinatorics*; Probability and Mathematical Statistics 17, Academic Press, New York, 1974.

[GJ79] Michael R. Garey, David S. Johnson: *Computers and Intractability — A Guide to the Theory of NP-Completeness*; W. H. Freeman and Company, New York, 1979.

[GMR88] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; *SIAM Journal on Computing 17/2* (1988) pp. 281-308.

[GMS74] E. N. Gilbert, F. J. Mac Williams, N. J. A. Sloane: Codes which Detect Deception; *The Bell System Technical Journal 53/3* (1974) pp. 405-424.

[GY89] Ronald L. Graham, Andrew C. Yao: On the Improbability of Reaching Byzantine Agreements; *Proc. 21st Symposium on Theory of Computing (STOC)*, ACM, New York, 1989, pp. 467-478.

[IR89] Russell Impagliazzo, Steven Rudich: Limits on the Provable Consequences of One-way Permutations; *Proc. 21st Symposium on Theory of Computing (STOC)*, ACM, New York, 1989, pp. 44-61.

[JS95] Thomas Johansson, Ben Smeets: On A^2-Codes Including Arbiter's Attacks; *Proc. Eurocrypt '94*, Lecture Notes in Computer Science 950, Springer-Verlag, Berlin, Germany, 1995, pp. 456-460.

[LSP82] Leslie Lamport, Robert Shostak, Marshall Pease: The Byzantine Generals Problem; *ACM Transactions on Programming Languages and Systems 4/3* (1982) pp. 382-401.

[P93] Birgit Pfitzmann: Sorting Out Signature Schemes; *Proc. 1st ACM Conference on Computer and Communications Security*, ACM, New York, 1993, pp. 74-85.

[P96] Birgit Pfitzmann: *Digital Signature Schemes: General Framework and Fail-Stop Signatures*; Lecture Notes in Computer Science 1100, Springer-Verlag, Berlin, Germany, 1996.

[PSL80] Marshall Pease, Robert Shostak, Leslie Lamport: Reaching Agreement in the Presence of Faults; *Journal of the ACM 27/2* (1980) pp. 228-234.

[PW92] Birgit Pfitzmann, Michael Waidner: *Unconditionally Untraceable and Fault-tolerant Broadcast and Secret Ballot Election*; Hildesheimer Informatik-Berichte 3/92, ISSN 0941-3014, University of Hildesheim, Institut für Informatik, 1992.

[R80] Michael O. Rabin: Probabilistic Algorithms in Finite Fields; *SIAM Journal on Computing 9/2* (1980) pp. 273-280.

[R90] John Rompel: One-Way Functions are Necessary and Sufficient for Secure Signatures; *Proc. 22nd Symposium on Theory of Computing (STOC)*, ACM, New York, 1990, pp. 387-394.

[R94] Tal Rabin: Robust Sharing of Secrets when the Dealer Is Honest or Cheating; *Journal of the ACM 41/6* (1994) pp. 1089-1109.

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

[S90] Gustavus J. Simmons: A Cartesian Product Construction for Unconditionally Secure Authentication Codes that Permit Arbitration; *Journal of Cryptology 2/2* (1990) pp. 77-104.

[W91] Michael Waidner: *Byzantinische Verteilung ohne kryptographische Annahmen trotz beliebig vieler Fehler* (in German); Dissertation, University of Karlsruhe, Fakultät für Informatik, 1991.

[WC81] Mark N. Wegman, J. Lawrence Carter: New Hash Functions and Their Use in Authentication and Set Equality; *Journal of Computer and System Sciences 22* (1981) pp. 265-279.