STACS '92 (9th Symposium on Theoretical Aspects of Computer Science), Paris, Feb. 13-15, 1992;
LNCS 577, Springer-Verlag, Berlin 1992, 339-350.

Unconditional Byzantine Agreement for any Number of Faulty Processors

— Extended Abstract —

Birgit Pfitzmann
Institut für Informatik
Universität Hildesheim
Samelsonplatz 1
W-3200 Hildesheim, Germany
pfitzb@infhil.uucp

Michael Waidner
Institut für Rechnerentwurf
und Fehlertoleranz
Universität Karlsruhe
Postfach 6980
W-7500 Karlsruhe, Germany
waidner@ira.uka.de

Abstract. We present the first Byzantine agreement protocol which tolerates *any* number of maliciously faulty processors *without* relying on computational assumptions (such as the unforgeability of digital signatures).

Our protocol needs reliable broadcast and secret channels in a precomputation phase. For a security parameter σ, it achieves Byzantine agreement with an error probability of at most $2^{-\sigma}$, whereas all computations are polynomial in σ and the number of processors.

The protocol is based on an unconditionally secure authentication mechanism, called pseudosignatures. Pseudosignatures are a generalization of a mechanism by CHAUM and ROIJAKKERS and might be useful in other protocols, too.

1 Introduction

Byzantine agreement protocols (BAPs) are an important primitive for distributed computations. They are intended to achieve reliable broadcast where this is not available physically and where some processors may be faulty. Correct agreement on the value $v \in D$ (where $|D| \geq 2$) of a transmitter means [PeSL_80]:

Consistency All good processors agree on the same value $v \in D$.

Correctness If the transmitter is good, v is the value it meant to send.

Let n be the number of processors and t an upper bound on the number of faulty processors. Faulty processors are assumed to be malicious. Except for preventing good processors from communicating, they can do whatever they like. In particular, it is *not* presupposed that faulty processors are limited to polynomial-time computations.

On the presupposition of an **authentication mechanism**, any $t < n$ can be tolerated [PeSL_80, DoSt_83] (**authenticated BAP**): Such a mechanism enables each good processor P to *authenticate* its messages so that any recipient R can locally verify whether a received message really comes from P. This must be possible even if R receives the message indirectly, via an arbitrary number of intermediate processors. Of course, maliciously faulty processors will try to forge such authentications.

Till now, there has been no *completely* verified authentication mechanism, thus no *completely* verified authenticated BAP: The most secure substitute were **digital signature** schemes [DiHe_76, GoMR_88], and all known schemes are based on *unproven* computational assumptions (e.g., "factoring is hard" [GoMR_88] or "one-way functions exist" [Romp_90], with respect to polynomially bounded faulty processors). But even if one of these computational assumptions could be proved (which would imply $\mathcal{P} \neq \mathcal{NP}$), digital signatures would still require that faulty processors are *polynomially* bounded.

A BAP which tolerates computationally *unbounded* faulty processors is called **unconditional**.

There was a folklore believe that unconditional BA can be achieved iff $t < n/3$. This was caused by [PeSL_80], which proved this impossibility result in a deterministic (i.e., errorfree) sense.

The first unconditional BAPs for numbers $t \geq n/3$ were presented in [BaPW_91, Waid_91: Ch. 2]: They tolerate any $t < n/2$ with an exponentially small error probability, i.e., $\leq 2^{-\sigma}$, in $O(\sigma)$ rounds [BaPW_91], or even in an expected constant number of rounds [Waid_91: Ch. 2.2.3]. All computations are polynomial in σ and n. It is presupposed that secret channels and reliable broadcast are available in a precomputation phase.

In the same model, we present the first unconditional BAP which tolerates *any* number $t < n$ (in $O(t)$ rounds).

Our BAP has an exponentially small error probability, too. One should not confuse this error probability with the computational assumptions of digital signatures: The former is a property which is rigorously proved, whereas the latter might be completely wrong.

Because of secret channels, faulty processors receive no information about the messages good processors exchange. For $t \geq n/3$ and an arbitrarily small error probability, secret channels are necessary. This can be derived from [GrYa_89].

Reliable broadcast in a precomputation phase is a presupposition for authenticated BAPs implemented with digital signatures, too: For such BAPs, all good processors must agree on the test keys of all processors. Thus, these test keys must be broadcast reliably before the agreement protocol can start. (Technically, the operators of the processors can perform the precomputation phase in a highly secure environment, e.g., with some other processors which are connected via a physically reliable broadcast network.)

The basis of our BAP is an unconditionally secure authentication mechanism, called **pseudosignatures** (Section 2), which is a generalization of a mechanism from [ChRo_90]. In Section 3 we obtain our BAP by fitting pseudosignatures into an authenticated BAP (that of [DoSt_83]).

Our constructions are described and proved in full detail in [Waid_91: Ch. 3].

Notations and assumptions: Let $P = \{P_1, \ldots, P_n\}$ be the set of processors, and $\sigma \in \mathbb{N}$ a security parameter so that an error probability of $2^{-\sigma}$ is acceptable. All our protocols are polynomial in σ and n.

If a statement is true with probability exponentially close to 1, we say that it holds **almost certainly**.

As usual with BAPs, we assume that each pair of good processors can communicate securely. We also assume that the network is synchronous. Additionally, we assume that in a *precomputation phase*, i.e., *before* the value which is to be distributed by a BAP is chosen,

- each processor can reliably broadcast some information, and
- each pair of processors has a secret and secure channel.

With these secret and secure channels, the assumption that each pair of good processors can communicate securely during the agreement protocol can be reduced to the assumption that faulty processors cannot prevent good processors from communicating [GiMS_74].

2 Pseudosignatures

Informally, pseudosignatures satisfy the following three properties:

Correctness If a good processor S pseudosigns a message, each good processor R would *accept* this message from S, almost certainly.

Unforgeability If a good processor R accepts a message from a good processor S, then S has pseudosigned this message, almost certainly.

Transferability For a given parameter $\lambda \in \mathbb{N}$, a message pseudosigned by a good S can be transferred λ times, e.g., via

$$S = P_{i_0} \to P_{i_1} \to \dots \to P_{i_\lambda} \quad ,$$

so that for each $j < \lambda$, if P_{i_j} accepts the message, then it knows that $P_{i_{j+1}}$ will accept the message, too, almost certainly.

If P_{i_λ} forwards the message, it is not guaranteed that the (λ+1)-st recipient accepts the message, too. Thus λ must be chosen carefully, depending on the specific protocol (e.g., if the protocol terminates deterministically in at most R rounds, $\lambda = R$ suffices).

For comparison, conventional digital signatures should satisfy these requirements, too, and should even be transferable an arbitrary number of times. But their unforgeability relies on unproven computational assumptions and they cannot withstand computationally unbounded adversaries (cf. Sect. 1).

In the following, we present unconditionally secure pseudosignatures. Our scheme is a generalization of the scheme from [ChRo_90], which dealt with the special case $\lambda = 2$.[1]

2.1 General structure

Conventional digital signatures cannot be unconditionally secure [DiHe_76]: The test keys must be public, i.e., a forger knows how a good processor P verifies the authenticity of a signed message. Thus, signatures can always be forged by brute force. (If testing a signature is in $\mathcal{P}$, then forging signatures is in $\mathcal{NP}$.)

For pseudosignatures, each processor P_i therefore needs a *different* test key, which is known only to P_i and to the pseudosigner, S [ChRo_90]. Thus, a *basic* pseudosignature ψ consists of n parts, called minisignatures; one for the test-key T_i of each P_i.

Forging a minisignature must be almost certainly impossible. This is achieved by using **authentication codes** [GiMS_74]: We use an **authentication key** as the test key T_i, and the encoding of the message v with T_i, $T_i(v)$, as the minisignature on v. There are efficiently computable authentication codes with a probability of successful forgery[2] of at most $2^{-\tau}$, where the key length is logarithmic in the length of v and linear in τ, and the length of each minisignature is τ+1 bit [WeCa_81]. For reasons of security, an authentication key can be used at most once.

---

1 In Section 5 of [ChRo_90], a scheme is sketched which should work for the general case $\lambda > 2$. Implicitly, this scheme assumes that the original sender of the message is good; thus this extension is not applicable here. (For readers familiar with [ChRo_90: Sect. 5]: The hash functions used there are useful only if they are *unknown* to the faulty processors.)

2 The probability of successful forgery is the probability that an adversary finds the correct value $T_i(v)$, provided it knows the value $T_i(v')$ for at most one $v' \neq v$.

Obviously, if S is faulty and knows which test keys belong to which processors, it can construct a basic pseudosignature which is acceptable for P_i, but not for P_j.

Therefore, the relation between test keys and processors is hidden: During an **initialization phase**, each processor chooses a test key randomly and sends it to S [ChRo_90]. This is done via a network which offers unconditional sender untraceability and perfect privacy. Thus, the pseudosigner only receives the *set* $\{T_1, \ldots, T_n\}$, and all other processors do not receive anything (Section 2.2).

Of course, this initialization does not prevent a faulty S from disturbing some minisignatures at random, i.e., from generating a basic pseudosignature which is acceptable for some processors, but not for all.

To prevent this attack, for each *complete* pseudosignature, m sets $\{T_{h,1}, \ldots, T_{h,n}\}$, $h = 1, \ldots, m$, are initialized independently, and the pseudosignature ψ on the message v consists of the m basic pseudosignatures $\psi_h := \{T_{h,1}(v), \ldots, T_{h,n}(v)\}$ [ChRo_90].

Now assume that a good processor P_i receives a pseudosignature $\psi = (\psi_1, \ldots, \psi_m)$ on a message v. Let a_i denote the number of different indices $h \in \{1, \ldots, m\}$ so that ψ_h contains a minisignature which passes the test with $T_{h,i}$.

If P_i receives (ψ, v) from S directly, $a_i = m$ must hold. Otherwise, S is faulty, and there is no need to accept anything from a faulty processor. Therefore, if $a_i = m$, P_i **1-accepts** (ψ, v).

If P_i receives (ψ, v) from another processor $P_j \neq S$ which has 1-accepted (ψ, v), we cannot necessarily expect $a_i = m$. If, e.g., P_i and P_j are the only good processors and S disturbs exactly one minisignature of a good processor, then either P_i or P_j 1-accept (ψ, v), but not both. Therefore, the acceptance rule is weakened for the second recipient. In [ChRo_90], P_i **2-accepts** (ψ, v) if $a_i \geq m/2$. Similarly, one can require $a_i \geq 1$: Then the probability that P_j 1-accepts (ψ, v) from S, and P_i does not even 2-accept (ψ, v), is at most 2^{-m}.

The scheme described so far is equivalent to the one proposed in [ChRo_90] (except that they only considered binary messages).

We extend this scheme from $\lambda = 2$ to arbitrary (but fixed) values of λ in a natural way: For a parameter $\Delta \in \mathbb{N}$ (which determines the error probability of transferability), we choose $m := (\lambda-1)\Delta+1$. If $a_i \geq m-(h-1)\Delta$, then P_i ***h*-accepts** (ψ, v), $h = 1, \ldots, \lambda$. For $h = 1$, this means $a_i = m$, for $h = \lambda$ this means $a_i \geq 1$.

In the following, we discuss the extended scheme in greater detail. We add some parts which were not completely described or proved in [ChRo_90], in particular, a provably correct initialization phase (Sect. 2.2). For all the three properties of pseudosignatures we prove upper bounds on the error probabilities, even against active attacks. This is especially important for transferability, where active attacks on recipients of pseudosignatures must be considered. Details omitted in this extended abstract can be found in [Waid_91: Ch. 3].

2.2 Initialization phase

During the initialization phase, the intended pseudosigner S is to receive the m sets $\{T_{h,1}, \ldots, T_{h,n}\}$, $h = 1, \ldots, m$. This is achieved by performing a **basic initialization protocol** m times in parallel. A similar initialization protocol is described in [ChRo_90], but without all necessary details, and without any proof of correctness.

The initialization phase needs secure and secret channels between each pair of processors and a reliable broadcast network. We say that a processor **publishes** a message M if M is sent via this reliable broadcast network.

Basis of the initialization protocol is the **DC-protocol** [Chau_88]:

Let $u \in \mathbb{N}$, and $F = GF(2^u)$, the finite field with 2^u elements. For each execution of the DC-protocol, each pair of processors $\{P_i, P_j\}$ needs a common secret DC-key, $K_{i,j} \in F$, and each processor P_i has a local input $x_i \in F$. In the following, x_i will be the encoding of an authentication key, thus u must be chosen sufficiently large (at least τ). Each processor P_i computes and publishes its local sum $O_i := x_i + K_{i,1} + \ldots + K_{i,n}$ and computes the global sum $\Sigma := O_1 + \ldots + O_n$ from all published local sums.

Obviously, Σ is equal to $x_1 + \ldots + x_n$, and one can prove that the ensemble of the local sums does not contain *any* information about the x_i's except Σ [Chau_88]. If S is not faulty and does *not* publish its local sum, even Σ is hidden from the faulty processors.

Thus, the DC-protocol enables S to receive the sum $\Sigma = x_1 + \ldots + x_n$, while no processor except S obtains any information about Σ, and not even S obtains any information about which x_i was chosen by which processor.

Finally, S should not receive the *sum*, but the *set* of all x_i's. To achieve this, we use a multiple-access protocol from [BoBo_90]: In n consecutive iterations of the DC-protocol, S is to receive the first n odd power sums of the x_i's, i.e., $x_1 + \ldots + x_n$, $x_1^3 + \ldots + x_n^3$, …, $x_1^{2n-1} + \ldots + x_n^{2n-1}$. If all processors behave well, then from these n power sums, S can compute the set of all x_i's which were chosen by an odd number of processors, almost certainly in polynomial time [Burt_71, Rabi_80: Th. 6].

If the faulty processors did not disturb the initialization, the basic initialization protocol would be trivial now: Each processor P_i would choose x_i as an encoding of a randomly chosen authentication key. Almost certainly, all good processors choose different keys. Thus S would obtain a set which contains at least the authentication keys of all good processors, almost certainly. However, faulty processors can easily disturb the initialization, e.g., by publishing wrong O_i's. Then, S receives garbage.

To handle this denial-of-service attack, S must be able to *detect* such disturbances, and it must be possible to *localize* disturbers:

To be able to *detect* disturbances, S is to receive the first $2n$ odd power sums, instead of the first n only. S uses the first n ones as described above and tests whether the second n ones are compatible with the set of the x_i's just computed. It can be seen that if the received sums pass this test, then almost certainly no disturbance has occured.

Disturbers are *localized* as described in [Chau_88]: After a detected disturbance, all data (i.e., messages published, DC-keys, local inputs x_i) are verified publicly. (N.B.: We presuppose reliable broadcast during this precomputation phase.) Each processor which behaves obviously badly is *eliminated*. If two processors P_i, P_j do not agree on their common DC-key $K_{i,j}$, it cannot be decided which of them was faulty. Thus only this disputed DC-key is *eliminated* once and for all (i.e., P_i and P_j never use a common DC-key again). If there are two groups of processors with no keys left between them, we define that these two groups perform the DC-protocol *independently*. After less than $n(n-1)/2$ detected disturbances, there are no keys left between good and faulty processors. Thus the good processors perform the DC-protocol by themselves and can never be disturbed again.

To initialize a complete pseudosignature, m basic initializations are performed. The result of this initialization is described by the following Theorem 1.

Theorem 1: (Correctness of pseudosignatures)

After the initialization phase, the following holds with probability $\geq 1-mn^4\, 2^{-\tau+1}$: If the pseudosigner S is good and forms the pseudosignature ψ on message v, then each good processor will 1-accept the pair (ψ, v). ♦

In the worst case, the initialization phase needs $2n^2$ rounds and messages of polynomial length. If w pseudosignatures are initialized sequentially, and if eliminations of DC-keys or of faulty processors are valid for all initializations, all w initializations together need at most $O(w+n^2)$ rounds.

2.3 Pseudosignatures with λ levels of acceptance

Pseudosignatures with λ levels of acceptance have already been defined in Section 2.1. It only remains to choose a value for Δ: To achieve an error probability for the transferability of at most $2^{-\kappa}$, we choose $\Delta := \lceil 2\,\lambda\kappa \ln 2 \rceil$.

For this parameter, we prove the following Theorem 2, which essentially says that pseudosignatures are unconditionally unforgeable and transferable. Theorem 2 considers passive attacks only (i.e., there is no interaction between good and faulty processors). Active attacks are considered in the second half of this section in general, and in Section 3 for a specific BAP.

Our definition of Δ implies $m \leq 2\lambda^2\kappa$, thus a pseudosignature ψ is of length $\leq 2n\lambda^2\kappa(\tau+1)$ bit.

Theorem 2: (Unforgeability and transferability in spite of passive attacks)

a) Assume S is good, and consider a good processor P and an arbitrary pair (ψ, v). Then P λ-accepts ψ as a pseudosignature on v with probability $\leq mn\, 2^{-\tau}$. This is true even if a correct pseudosignature (ψ', v') on a message $v' \neq v$ is known.

b) Let P and Q be two good processors, and $k < \lambda$. Then for each pair (ψ, v), the probability that P k-accepts, while Q not even $(k+1)$-accepts, is at most $2^{-\kappa}$. ♦

Sketch of proof

a) Follows from the unforgeability of authentication codes (Sect. 2.1).

b) We consider the worst case only: P and Q are the only good processors. Thus S is faulty, and for each of the m parts of the pseudosignature, the faulty processors know which two authentication keys belong to P and Q. But they do not know which key was chosen by P and which by Q. For each of the m parts, the faulty processors can decide to change none, one, or two of the two correct minisignatures. For reasons of symmetry, we can describe this decision by just two parameters:

g denotes the number of parts where both minisignatures are changed.

h denotes the number of parts where exactly one minisignature is changed.

Let $x := (k-1)\Delta$; this is the maximum number of minisignatures for P which may be changed so that P still k-accepts ψ. Let B_h denote the binomial distribution with parameter h and probability 1/2, i.e.,

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

With probability $\binom{h}{y} 2^{-h}$, within the h parts where one minisignature was changed, exactly y changed minisignatures belong to P. The faulty processors are successful if they have changed at most x minisignatures for P and more than $x+\Delta$ minisignatures for Q, altogether, i.e., $y \leq \min\{x-g, g+h-x-\Delta-1\}$.

Therefore, the faulty processors are successful with probability

$$P(g, h) := B_h(\min\{x{-}g,\, g{+}h{-}x{-}\Delta{-}1\}) \le B_h\Big(\frac{(x{-}g) + (g{+}h{-}x{-}\Delta{-}1)}{2}\Big) = B_h\Big(\frac{h}{2} - \frac{\Delta{+}1}{2}\Big). \qquad (✺)$$

For each $\gamma \ge 0$, it follows from [ErSp_74: (3.8)] that

$$B_h\Big(\frac{h}{2} - \gamma - \frac{1}{2}\Big) \le \sum_{z\, <\, h/2-\gamma} \binom{h}{z} 2^{-h} < \exp\Big(-\frac{2\gamma^2}{h}\Big).$$

We apply this to (✺). We want to show that $P(g, h) \le 2^{-\kappa}$.

$$P(g, h) \le B_h\Big(\frac{h}{2} - \frac{\Delta}{2} - \frac{1}{2}\Big) < \exp\Big(-\frac{\Delta^2}{2h}\Big).$$

Thus, $P(g, h) \le 2^{-\kappa} \Leftarrow \Delta^2 \ge 2h\, \kappa \ln 2$. With $m \ge h$, we obtain

$$P(g, h) \le 2^{-\kappa} \Leftarrow \Delta^2 \ge 2m\, \kappa \ln 2 = 2\,\big((\lambda{-}1)\Delta + 1\big)\, \kappa \ln 2,$$

and one can easily verify that the last inequality is satisfied if $\Delta \ge 2\, \lambda\kappa \ln 2$. ❐

Theorem 2 deals with passive, i.e., non-interactive, attacks only. In general, a protocol enables interactive attacks: After an attack, the faulty processors receive some information about how successful their attack was. Depending on this information, they can repeat their attack.

As an example, consider the following attack on a recipient of a pseudosignature: Let ψ be a correct pseudosignature, and let P be a good processor. For each interaction, the faulty processors change exactly one minisignature in ψ and send the result to P, which answers with its level of acceptance. Processor P 1-accepts iff none of its minisignatures was changed. Therefore, after a certain number of interactions, the faulty processors know exactly which minisignatures belong to P. Hence, they can change ψ so that P will 1-accept the changed ψ, whereas all other good processors will not even λ-accept it.

In order to limit the success probability of an active attack, the lenght or kind of the attack must be restricted. We do this by assuming that pseudosignatures are used in a *deterministically polynomial* protocol (polynomial in n and a security parameter ω). Thus, the faulty processors can repeat their attack a polynomial number of times, u. We also assume that the security parameters τ, κ can be chosen arbitrarily. ([Waid_91: Sect. 3.1.6] contains a more formal description of this.) In this case, we can estimate the probability of success of the faulty processors by brute force:

First, consider unforgeability: In up to u iterations, the faulty processors select a pair (ψ, v), send it to P, and receive either "λ-accepted" or "rejected". They are successful if P λ-accepts at least one of these pairs. Applying Theorem 2a to all the u pairs yields a success probability of at most $umn\, 2^{-\tau}$. If we choose $\tau := \lceil \omega + \log_2(umn) \rceil$, this is at most $2^{-\omega}$.

Secondly, consider transferability: Here, the faulty processors send at most u pairs to P and at most u to Q. Each time, they receive one out of at most $\lambda{+}1$ different reactions ("k-accepted" for $k = 1, \ldots, \lambda$, or "rejected"). Without loss of generality, their strategy is deterministic[3] and can be described by a *strategy tree*: Each node contains a pair (ψ, v) and the processor $X \in \{P, Q\}$ which receives the pair. Each child of a node belongs to exactly one of the $\lambda{+}1$ possible reactions of X. The depth of the tree is at most $2u{-}1$, and therefore it contains at most $\lambda^{-1}(\lambda{+}1)^{2u+1}$ nodes, i.e., pseudosignatures. Applying Theorem 2b to each of these pseudosignatures yields an error probability of at most $\lambda^{-1}(\lambda{+}1)^{2u+1}\, 2^{-\kappa}$. (Note that whether a pseudosignature is transferable from P to Q is statically fixed in each event; the

---

3 If they use an indeterministic strategy, then for each indeterministic decision there is one with maximum probability of success. If they always choose this one, their strategy remains at least equally successful.

whole information from the active attack lies in the decisions whether this node is reached at all in an actual protocol run.) Of course, the strategy tree contains an exponential number of nodes, but this influences the security parameter κ only logarithmically: If we choose $\kappa := \lceil \omega + (2u+1)\log_2(\lambda+1) - \log_2(\lambda) \rceil$, the success probability is at most $2^{-\omega}$.

In Section 3, we consider a specific BAP. In order to obtain more precise upper bounds on the error probabilities, we define the strategy tree specifically for this protocol and count the nodes more precisely.

3 Byzantine Agreement

In the following, we modify a well-known authenticated BAP [DoSt_83: Th. 3] so that it can be implemented with pseudosignatures instead of conventional digital signatures (Sect. 3.1), and we prove that it achieves BA almost certainly (Sect. 3.2). At the end, we sketch how we can use a fixed number of initialized pseudosignatures to perform an arbitrary number of BAs (Sect. 3.3).

3.1 Byzantine agreement protocol with pseudosignatures

The following BAP is based on the authenticated BAP from [DoSt_83: Th. 3]; for each "signature", we have determined the level of acceptance, and we have kept the strategy tree described in Section 2.3 as small as possible.

Let σ be the security parameter of the protocol, i.e., we accept an error probability of at most $2^{-\sigma}$, and let $t < n$ be an upper bound on the number of faulty processors.

The protocol uses pseudosignatures with $\lambda = t+1$ levels of acceptance with parameters $\kappa := \lceil \sigma + 18 \log_2(n) + \log_2(72) \rceil$ and $\tau := \lceil \sigma + \log_2(\kappa) + 7 \log_2(n) + \log_2(24) \rceil$. The length of the messages which must be pseudosigned is equal to the length of the transmitter's message.

Each execution of the BAP needs its own precomputation phase, where $2n-1$ pseudosignatures are initialized[4]. The transmitter T acts as the pseudosigner for one of them, each other processor for two of them. We call the two pseudosignatures of $P \neq T$ its **A-signature** and **B-signature**, and the pseudosignature of T its **A-signature**. Pseudosignatures will only be used in triples (i, α, ψ), where i denotes the index of the supposed pseudosigner P_i, $\alpha \in \{A, B\}$ the "type", and ψ the pseudosignature itself. A processor P ***k*-accepts** a triple (i, α, ψ) as a pseudosignature on v if it k-accepts (ψ, v) with respect to the initialization denoted by (i, α).

Protocol messages always contain a value v and a set of up to $t+1$ triples (i, α, ψ). For each level of acceptance $k \in \{1, \ldots, t+1\}$, we call a protocol message M ***k*-consistent**[5] for P_i iff M is of the form $(v, \{(i_1, \alpha_1, \psi_1), \ldots, (i_k, \alpha_k, \psi_k)\})$ and

- according to the parameters i_j, M contains k pseudosignatures of k different processors,
- according to the parameters i_j and α_j, M contains the A-signature of the transmitter,

---

4 In Section 3.3, we sketch how the BAP can be executed u times with a number of initialized pseudosignatures independent of u.

5 In [DoSt_83], k-consistent protocol messages are defined in a slightly different way: There, each processor signs not just v, but the whole protocol message. Our definition results in smaller messages to be signed and does not influence the fault tolerance of the protocol.

- P_i k-accepts all the triples (i_j, α_j, ψ_j) as pseudosignatures on v.

P_i **relays** a protocol message M containing v in the following way:

- If M is the first message P_i relays, P_i pseudosigns v using its A-signature, otherwise using its B-signature. P_i adds the resulting triple (i, α, ψ) to M.
- P_i forwards M (including its own triple (i, α, ψ) now) to each processor of whom no pseudosignature is contained in M yet.

During the protocol, each processor P_i has to relay at most two protocol messages, which are k-consistent for him.

The set $\boldsymbol{ACC_i}$ contains all the values which P_i has accepted from the transmitter. At the end, if ACC_i contains exactly one value v, P_i accepts this value as its **final value**. Otherwise, P_i knows that the transmitter was faulty. The set $\boldsymbol{OLD_i}$ ensures that a good processor P_i reacts to at most two protocol messages from each processor P_x. This keeps the strategy tree relatively small (see Sect. 3.2).

A good transmitter T starts with a value v, which is to be distributed.

Byzantine Agreement Protocol:

[0] Transmitter T relays v, i.e., T pseudosigns v with its A-signature and forwards it to all the other processors. Each $P_i \neq T$ initializes two empty sets OLD_i and ACC_i.

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

[k] a. P_i forms the set $N_{i,k}$ of *new* protocol messages:
For each protocol message M that P_i received in round [k–1]: Let P_x be the sender of M. If M fulfills the first and second condition of k-consistency and contains a triple (x, α, ψ) and $(x, \alpha) \notin OLD_i$, then P_i adds M to $N_{i,k}$ and (x, α) to OLD_i. Otherwise P_i ignores M. (In this case, P_i knows that P_x is faulty, and there is no need to consider messages from faulty processors.)

b. P_i forms the set $V_{i,k}$ of all k-consistent messages from $N_{i,k}$. Let $V_{i,k}$ be lexicographically ordered. P_i adds all the values which are included in a message in $V_{i,k}$ to ACC_i.

c. If $k \leq t$ and $V_{i,k} \neq \emptyset$, and if P_i has not relayed any message before, then P_i *relays* the first two messages in $V_{i,k}$ which contain different values, or just the first one, if all messages contain the same value.

d. If $k \leq t$ and $V_{i,k} \neq \emptyset$, and if P_i relayed exactly one message, which contained a value v', then P_i *relays* the first message in $V_{i,k}$ which contains a value $v'' \neq v'$, if there is one.

[end] The **final value** of T is its own local input v. If ACC_i contains exactly one value v, processor P_i takes this v as its final value; otherwise, P_i decides on "faulty transmitter". ♦

3.2 Proof of the Byzantine agreement protocol

The correctness of the protocol is proved in two steps:

Lemma 3 states that if pseudosignatures were ideally secure, then our BAP would achieve errorless BA. Theorem 4 states that the assumption made in Theorem 3 holds with probability $\geq 1–2^{-\sigma}$, i.e., the error probability of our BAP is at most $2^{-\sigma}$.

Lemma 3: (Deterministic version of Theorem 4)

For any $t < n$, the protocol described above achieves Byzantine agreement if for all pseudosignatures ψ of the protocol run, all $\alpha \in \{A, B\}$, all good processors P and Q, all values v, and all rounds $[k]$ the following is true:

A_1 If ψ is the α-signature of P on v, then Q 1-accepts ψ.

A_2 If in round $[k]$, P accepts ψ as an α-signature of Q on v, then Q has relayed a protocol message containing the α-signature of Q on v not later than in round $[k-1]$.

A_3 If P k-accepts ψ as an α-signature of some processor R, then Q would $(k+1)$-accept ψ. ♦

Lemma 3 and its proof are very similar to [DoSt_83: Th. 3]; thus we omit the proof.

Theorem 4: (Byzantine agreement)

For any $t < n$, the protocol described above achieves Byzantine agreement with an error probability of at most $2^{-\sigma}$. ♦

<u>Sketch of proof</u>

Let W_i be the probability that assumption A_i of Lemma 3 does not hold. Then, the error probability of the protocol is at most $W_1 + W_2 + W_3$.

Estimation of W_1: From Theorem 1, we know the probability corresponding to W_1 for each of the $\leq 2n$ pseudosignatures, and it is easy to see that $m \leq 2\lambda^2\kappa$. Thus we multiply the probability of Theorem 1 by $2n$, apply $\lambda \leq n$, and obtain $W_1 \leq \kappa n^7\, 2^{-\tau+3}$.

Estimation of W_2: In a similar way, we obtain $W_2 \leq \kappa n^6 2^{-\tau+3}$.

Estimation of W_3: Let W_3^* be the probability corresponding to W_3 for *one* of the $\leq n^2/2$ pairs $\{P, Q\}$ and *one* of the $\leq 2n$ pseudosignatures (i, α) initialized. Thus we have $W_3 \leq n^3 W_3^*$. (We need not consider the λ different levels of acceptance, since in each round $[k]$, P or Q either k-accept a pseudosignature or reject it.)

As in Section 2.3, we estimate W_3^* by considering an overly pessimistic case: P and Q are the only good processors, and the faulty processors know everything, except which of the two authentication keys of P and Q belong to P and which to Q, for the m parts of the pseudosignature considered. Again, we count pseudosignatures in a strategy tree B.

For our specific BAP, we define B slightly differently: Each node of B represents one round and is described by the protocol messages that P and Q received in that round. Since P and Q do not react to old messages (Step $[k.a]$), it is sufficient to consider *new* messages only.

The depth of B is t $(\leq n)$, corresponding to the rounds $[0]$ ($\approx$ root), …, $[t]$ ($\approx$ leaves) of the BAP. For each node of depth $k < t$, P and Q can react in round $[k+1]$ by relaying up to two of the new messages just received. Each child corresponds to one reaction and can be described unambiguously by specifying which messages P and Q relay. (The pseudosignature that P or Q adds to a message before relaying it is already known.) Call the child corresponding to the case that neither P nor Q relays a message the *leftmost* child.

During the protocol, each of P and Q considers at most two protocol messages from each of the $< n$ faulty processors, and each protocol message contains the pseudosignature we are interested in at most once. Therefore, the messages in each path in B contains $< 4n$ pseudosignatures. If w denotes the number of different paths in B, then, according to Theorem 2b, $W_3^* \leq w\ 4n\ 2^{-\kappa}$.

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

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

different possibilities for P and Q each. Therefore, until the first branch to the right, a node has $\le x_0 := 4n^4$ 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 $\le x_1 := 4n^3$ possibilities left, after the second, $\le x_2 := 4n^2$, after the third, $\le x_3 := 2n$, and of course, after the fourth, $x_4 = 1$. In this way, we obtain an upper bound on w: Each path can be uniquely described by specifying in which of the $n-1$ possible depths the $L \le 4$ branches to the right occur, and for the i-th branch ($i = 1, \ldots, L$), which of the $\le x_{i-1} - 1$ branches to the right is chosen. Thus, there are

$$w_L \le \binom{n}{L} x_0 \ldots x_{L-1}, \; L = 0, 1, 2, 3, 4,$$

paths with exactly L branches to the right. After some computation, this yields $w \le 9n^{14}$.

Thus, we obtain $W_3^* \le w\ 4n\ 2^{-\kappa} \le 36\ n^{15}\ 2^{-\kappa}$ and $W_3 \le n^3\, W_3^* \le 36\ n^{18}\ 2^{-\kappa}$.

Finally, this implies

$$W \le W_1 + W_2 + W_3 \le \kappa\,(8n^7 + 8n^6)\, 2^{-\tau} + 36\ n^{18}\ 2^{-\kappa},$$

and one can easily verify that both summands are at most $2^{-\sigma-1}$. ❐

3.3 Fixed use of the reliable broadcast network

The BAP of Section 3.1 needs $2n-1$ pseudosignatures for each execution. This suggests that the number of pseudosignatures initialized during the precomputation phase, i.e., using a physically reliable broadcast network, must be proportional to the intended number of protocol executions.

However, using a bootstrapping mechanism, we need to prepare just $O(n^2)$ executions in order to execute the protocol u times, for arbitrary u.

The parameter u influences the protocol performance via the security parameter σ only: σ must be logarithmic in u. Obviously, this is necessary, because the error probability of u executions is about u times the error probability of one execution. Therefore, we need $\sigma \ge \omega + \log_2(u)$ to achieve an error probability of at most $2^{-\omega}$.

Thus we have a similar situation as with conventional digital signatures.

The idea is the following:

Assume that in the precomputation phase, we have initialized a certain number of pseudosignatures. If the BAP must be executed, we first initialize the necessary $2n-1$ pseudosignatures. Unfortunately, at this time we cannot access the physically reliable broadcast network any longer. (Otherwise, Byzantine agreement would be trivial.) Thus, we use our BAP to implement reliable broadcast, i.e., we consume some of our already initialized pseudosignatures. To be able to repeat this procedure, we must initialize some additional pseudosignatures to replace the consumed ones. This is possible because with a fixed number of reliably broadcast messages, we can initialize as many pseudosignatures as we like.

4 Summary

In Section 2, we showed how pseudosignatures with an arbitrary number of levels of acceptance can be implemented. We also demonstrated that, even though active attacks must be considered, these pseudosignatures can be an unconditionally secure substitute for conventional digital signatures in a large class of deterministically polynomial protocols.

In Section 3, we presented a modification of a well-known authenticated Byzantine agreement protocol (BAP) and proved that it can be implemented securely with pseudosignatures. This results in the first BAP tolerating *any* number $t < n$ of faulty processors without relying on unproven computational assumptions.

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

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

References

BaPW_91 Birgit Baum-Waidner, Birgit Pfitzmann, Michael Waidner: Unconditional Byzantine Agreement with Good Majority; STACS '91, LNCS 480, Springer-Verlag, Heidelberg 1991, 285-295.

BoBo_90 Jurjen Bos, Bert den Boer: Detection of disrupters in the DC protocol; Eurocrypt '89, LNCS 434, Springer-Verlag, Berlin 1990, 320-327.

Burt_71 Herbert O. Burton: Inversionless Decoding of Binary BCH Codes; IEEE Transactions on Information Theory 17/4 (1971) 464-466.

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

ChRo_90 David Chaum, Sandra Roijakkers: Unconditionally Secure Digital Signatures; Crypto '90, Santa Barbara, 11-15 August 1990, Abstracts, 209-217.

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

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

ErSp_74 Paul Erdös, Joel Spencer: Probabilistic Methods in Combinatorics; Probability and Mathematical Statistics 17, Academic Press, New York 1974.

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

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

GrYa_89 Ronald L. Graham, Andrew C. Yao: On the Improbability of Reaching Byzantine Agreement; 21st STOC 1989, ACM Press, New York 1989, 467-478.

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

Rabi_80 Michael O. Rabin: Probabilistic Algorithms in Finite Fields; SIAM J. Comput. 9/2 (1980) 273-280.

Romp_90 John Rompel: One-Way Functions are Necessary and Sufficient for Secure Signatures; 22nd STOC 1990, ACM Press, New York 1990, 387-394.

Waid_91 Michael Waidner: Byzantinische Verteilung ohne kryptographische Annahmen trotz beliebig vieler Fehler; Universität Karlsruhe, Fakultät für Informatik, Dissertation, October 1991; to appear.

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