Document #	434SR011	Activity Paper	
Title	Optimal Efficiency of Optimistic Contract Signing		
Author/Body	B. Pfitzmann (SRB), M. Schunter (UDO), M. Waidner (ZRL)		
Editor	M. Schunter (UDO)		
Reviewer	M. Steiner (ZRL)		
Date	04/20/98		
Status	Draft	Vers. 1	SEMPER internal
CEC/TA id	none		

ACTIVITY PAPER

434SR011
04/20/98

VERSION 1

A shorter version of this report was published as: Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Optimal Efficiency of Optimistic Contract Signing; 17th Symposium on Principles of Distributed Computing (PODC), ACM, New York 1998, 113-122.

Optimal Efficiency of Optimistic Contract Signing

B. Pfitzmann (SRB), M. Schunter (UDO), M. Waidner (ZRL) *

SEMPER Activity Paper 434SR011
Draft Vers. 1 (SEMPER internal)
04/20/98

ABSTRACT

A contract is a non-repudiable agreement on a given contract text, i.e., a contract can be used to prove agreement between its signatories to any verifier. A contract signing scheme is used to fairly compute a contract so that, even if one of the signatories misbehaves, either both or none of the signatories obtain a contract.

Optimistic contract signing protocols use a third party to ensure fairness, but in such a way that the third party is not actively involved in the fault-less case. Since no satisfactory protocols without any third party exist, this seems to be the best one can hope for.

We prove tight lower bounds on the message and round complexity of optimistic contract signing on synchronous and asynchronous networks, and present new and efficient protocols based on digital signatures which achieve provably optimal efficiency.

Furthermore, we investigate what can be gained if the third party participates in the contract verification.

Table of Contents

1 Introduction **3**
1.1 Related Work 3
1.2 Our Results 4

2 Definitions **6**
2.1 Network Models and Protocol Complexity 6
2.2 Contract Signing 7
2.3 Optimistic Contract Signing 8
2.4 Notations and Assumptions 9

3 A Message-Optimal Synchronous Scheme **10**

4 A Round-Optimal Synchronous Scheme **13**

5 A Time-Optimal Asynchronous Scheme **14**

6 An Optimal Asynchronous Non-Optimistic Scheme **19**

7 An Optimal Synchronous Scheme with Three-Party Verification **21**

8 A Optimal Asynchronous Scheme with Three-Party Verification **24**

9 Conclusion **27**

* Edited by M. Schunter (UDO)

10 Acknowledgment **27**

1 Introduction

A *contract* is a non-repudiable agreement on a given text [Blum 81]. A contract signing scheme includes at least three players and two protocols: Two signatories participate in a contract signing protocol "`sign`" which fairly computes a contract. This contract can then be used as input to a contract verification protocol "`show`" to convince any verifier such as a court that the signatories reached agreement on the given text.

Note that unlike cryptographic contract signing protocols [Blum 81], our notion does not tolerate uncertainty about the outcome. In the end, the user must have a definitive answer whether a valid contract was produced or not. Furthermore, we achieve deterministic fairness if the underlying digital signature scheme is secure.

In all practical schemes, contract signing involves an additional player, called third party. This party is (at least to some extent) trusted to behave correctly, thus playing the role of a notary in paper-based contract signing. A well-known protocol for contract signing by exchanging signatures via a third party works as follows (see also Scheme 4): Both signatories send their signatures to the third party. The third party then verifies and forwards them. At the end, both signatories end up having two signatures on the contract which may be sent to any verifier for verification. In this and similar protocols, the third party has to be involved in *all* executions of the contract signing protocol.

In order to minimize this involvement while guaranteeing fairness, the concept of so called "optimistic" protocols has been introduced [AsSW 97, BGMR 90][1]. The basic idea of optimistic schemes[2] is that the third party is not needed in the fault-less case: After the execution of the optimistic signing protocol, two correct signatories always end up with a valid contract. Only if one of the signatories misbehaves, the third party is involved to decide on the validity of the contract.

1.1 Related Work

The term "contract signing" was first introduced in [Blum 81]. In [EvYa 80], it was shown that no deterministic contract signing scheme (called "public-key agreement system" in [EvYa 80]) without third party exists if the verifier is state-less and only the two signatories participate in the contract signing protocol.

Contract signing without third party: Early research focused on probabilistic contract signing schemes based on *gradual exchange of signatures* [Blum 81, Blu2 83, EvGL 85, Gold 83] (see [Damg 95] for recent results): Both signatories exchange signatures "bit-by-bit." If one signatory stops prematurely, both signatories have about the same fraction of the peer's signature, which means they can complete the contract offline by investing about the same amount of work.

As pointed out in [BGMR 90], this approach is not satisfactory in real life: Consider, for example, a house selling contract. If the protocol stops prematurely, the seller cannot be sure whether the buyer will invest some years to complete the contract or not, i.e., whether the seller still owns the house and can look for another buyer or not. Thus, the seller is actually forced to take a high risk, or to complete the contract.

Contract signing with third party: Simple schemes for contract signing use an *online* third party, i.e, one that is actively involved in each run.

Optimistic contract signing with third party: The first somewhat[3] *optimistic* scheme has been described in [Even 83]. The first optimistic scheme in our sense is based on *gradual increase of privilege* [BGMR 90]: In n message exchanges, the probability with which a contract is valid is gradually increased from 0 to 1. If the protocol stops prematurely, each

[1] This paper includes the first author's notes on contract signing that were referred to in [AsSW 97].

[2] See also [BüPf 89, BüPf 90] for optimistic protocols for payment for receipt or goods, or [Mica 97, ZhGo 97] for recent optimistic protocols for certified mail, i.e., a fair exchange of a message for a signature on a receipt.

[3] It assumes that verification is a three-party protocol, i.e., that the contract is not valid on its own but only if a third party called "center of cancellation" does not object.

signatory can invoke a third party called "judge." The third party will wait until the protocol would have terminated (i.e., we are in a synchronous model). After this timeout, the third party picks a random value ρ in the interval $[0, 1]$, or retrieves it in case the third party was invoked for this contract before. If the probability given by the last message received by the invoking party is at least ρ, the contract is considered valid and an affidavit is issued and sent to both signatories. Otherwise the contract is considered invalid. By construction, if the protocol is prematurely stopped, one party might be "privileged", i.e., has a slight advantage when invoking the judge: If the third party chooses a ρ that lies between the probabilities of the two signatories, only one of them can finalize the contract. Thus, if a correct player A invokes the third party and gets the answer that the contract is invalid, she cannot be sure that the same would happen if B invokes the third party, i.e., that the contract is indeed not signed. In the worst case, B might have a valid contract (i.e., probability 1) and hence knows that if A complains, it will succeed only with the probability contained in the last message sent by B. The probability that such an uncertain situation arises is non-negligible, but linearly small in the number of messages exchanged [BGMR 90]. In the house-selling example mentioned above, such a non-negligible error would probably not be acceptable for the seller.

Recent research concentrated on optimistic contract signing schemes that avoid such uncertain situations, and guarantee a *definite* decision within limited time: [AsSW 97] describes a synchronous contract signing protocol with four messages. This was improved in [AsSW3 97] to a four-message protocol for asynchronous networks. Compared to this earlier work, the focus of this paper is on proving bounds on the message- and time-complexity of optimistic contract signing protocols for different models[4].

Commit Protocols: Compared to commit-protocols [SiKS 97] for atomicity of distributed transactions, contract signing aims at a non-repudiable agreement while assuming a byzantine failure model, i.e., even if most signatories are malicious, contract signing guarantees a correct outcome for correct signatories whereas commit-protocol do not.

Agreement Protocols: Contract signing achieves more than just agreement [PeSL 80]: besides reaching agreement, the players also want to be able to prove it afterwards.

Remark: Fair exchange of signatures[5] and *fair contract signing* are different problems since contract signing does not require a contract to be a text and two signatures. Obviously, contract signing can always be implemented based on fair exchange of signatures, but not all contract signing schemes exchange signatures. They only guarantee non-repudiation of the agreement on a contract.

1.2 Our Results

We present new and efficient optimistic contract signing schemes and prove that their efficiency with respect to messages or time is optimal if the signatories are correct and agree on the contract. Furthermore, we prove some bounds and limitations on optimistic contract signing in general. All our schemes are based on an arbitrary digital signature scheme. Tables 1 and 2 give a detailed summary of our results: We present a message- (3 messages) and a round-optimal (2 rounds) synchronous optimistic contract signing scheme as well as a time-optimal (time 3) asynchronous scheme. We prove the optimality of these new schemes as well as the optimality of the scheme described in [AsSW3 97] by proving tight bounds on message and time complexity of synchronous and asynchronous optimistic contract signing. Furthermore, we show that each message/time-optimal protocol is optimal with respect to time/messages given the message/time limitation. In Theorem 4 we show that no asynchronous optimistic contract signing scheme with state-less third party exists.

Finally, we compare the efficiency of the optimistic protocols with the efficiency of a well-known asynchronous non-optimistic scheme (4 messages in time 2) under the assump-

[4] For similar research on authentication protocols see [Gong2 95].

[5] Each player A receives a digital signature $\mathrm{sig}_{\mathsf{B}}(C)$ if and only if the other signatory B receives $\mathrm{sig}_{\mathsf{A}}(C)$, too.

Model			Our Results			
C	T	Op	$t^{(+)}$	$m^{(+)}$	Optimal	Proof
s	sl	+	3	$\underline{3}$	Scheme 1	Theorem 1
s	sl	+	$\underline{2}$	4	Scheme 2	Theorem 1
a	sl	+	Impossible			Theorem 4
a	sk	(+)	4	$\underline{4}$	See [AsSW3 97]	Theorem 2
a	sk	(+)	$\underline{3}$	6	Scheme 3	Theorem 3
a	sk	–	$\underline{2}$	$\underline{4}$	Scheme 4	Theorem 5
s	d, sk	(+)	$\underline{1}$	$\underline{2}$	Scheme 5	Theorem 6
a	d, sk	(+)	3	$\underline{3}$	Scheme 6	Theorem 7

Legend:

C	Communication Model: “s” for synchronous, “a” for asynchronous.
T	Properties of the third party: “sl” for state-less, “sk” for state-keeping, “d” if the third party is required to participate in verification.
Op	“+” stands for optimistic protocols (Def. 5), “(+)” stands for optimistic on agreement (Def. 6), and “–” stands for non-optimistic protocols (Def. 3).
$t^{(+)}$	Time for the output of a contract if the signatories agree (underlined figures are provably optimal, which is proven in the mentioned Theorem).
$m^{(+)}$	Number of messages in case of agreement.

Table 1: Provably Optimal Schemes By Model.

Theorem	Model			Result	
No.	C	T	Opt	$t^{(+)}$	$m^{(+)}$
1	s	sk	(+)		≥ 3
1	s	sk	(+)	$\geq 3 \leftarrow$	$= 3$
1	s	sk	(+)	≥ 2	
2	a	sk	(+)		≥ 4
3	a	sk	(+)	≥ 3	
3	a	sk	(+)	$= 3 \rightarrow$	≥ 6
4	a	sl	(+)	Does not exist	
5	a	sk	-		≥ 4
5	a	sk	-	≥ 2	
6	s	d, sk	(+)		≥ 2
7	a	d, sk	(+)		≥ 3
7	a	d, sk	(+)	$\geq 3 \leftarrow$	$= 3$

Table 2: Our Theorems and What They Prove (Legend: See Table 1).

tion that both signatories agree and do not misbehave. This shows that optimistic protocols achieve similar efficiency without interacting with a third party.

After these results for optimistic contract signing, we investigate what changes result from allowing the third party to participate in the verification protocol. We show that the optimistic scheme from [Even 83] with 1 round and 2 messages is optimal on a synchronous network whereas on an asynchronous network, one needs three messages in three rounds.

2 Definitions

2.1 Network Models and Protocol Complexity

We distinguish between the "standard" synchronous and asynchronous network models [Lync 96, Tel 91]. On synchronous networks, messages are guaranteed to be delivered within a so-called "round", i.e., a recipient of a message can decide whether a message was sent or not. This cannot be decided on asynchronous networks since messages may be delayed and reordered arbitrarily. For machines, we assume a byzantine failure model, i.e., a faulty machine may send arbitrary messages but must not be able to prevent delivery of messages between two correct machines. The time-complexity of a synchronous protocol is the number of rounds required for its execution. The time-complexity of an asynchronous protocol is the time required for its execution if transmission of each message requires time 1 and local computations require no time.

We assume that the network is reliable, i.e., that all messages sent between correct machines are eventually delivered. In asynchronous networks there is no notion of global time, no time-limit on the time needed for message transmission, and there is no guarantee that messages are delivered in the same order they were sent.

For both network types, we assume that each algorithm receives its messages from other algorithms and its local inputs, then does a computation on them and outputs at most one local output and one message for each other algorithm.

The time-complexity sketched above is formalized by defining a logical time [Lync 96]:

Definition 1 (Time Complexity)
The time complexity of a protocol is defined to be the highest clock assignment at the end of the protocol obtained by the following rules:

1. Each machine participating in the protocol has a time assignment $time \in IN$ and a mode assignment $mode \in \{\texttt{send}, \texttt{receive}\}$. In `send`-mode, messages can only be sent. In `receive`-mode, messages can only be received. Initially, $time := 0$ and $mode := \texttt{receive}$ is assigned.
2. The $time$ assignment of an algorithm is increased whenever an event happens.
   Unlike Lamport's time-stamps [Lamp 78], an event here is defined as "changing from receive-mode to send-mode". Consecutive send or receive operations as well as changes from send to receive mode do not change the local clock.
3. The assigned $time$ of the local clock of the sender is assigned to every message sent.
4. Whenever a time-stamp higher than the local $time$ assignment has been assigned to a received message, the local $time$ assignment is set to the time assigned to the received message.

□

For synchronous communication, we make the additional assumptions that there is a global notion of rounds and each message sent in round i by an correct player is delivered as input to round $i + 1$. This enables a receiver to decide whether a message has been sent or not. We assume that messages which do not arrive in their designated round are ignored. Note that in the synchronous model, the time-complexity defined by Definition 1 equals the minimum number of rounds needed for the protocol.

2.2 Contract Signing

We now give a formal definition of contract signing and describe the requirements we want to achieve.

Definition 2 (Contract Signing Scheme)
A *contract signing scheme* for a message space M and a set of transaction identifiers $TIDs$ is a triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ of probabilistic interactive algorithms (such as probabilistic interactive Turing Machines) where V is state-less, i.e., has no memory between subsequent protocol runs. The algorithms A and B are called signatories, and V is called verifier. The algorithms can carry out two interactive protocols:

Contract Signing (Protocol "`sign`*"):* Each signatory $\mathsf{X} \in \{\mathsf{A}, \mathsf{B}\}$ obtains a local input $(\texttt{sign}, C_X, tid)$, where `sign` indicates that the "`sign`"-protocol shall be executed, $C_X \in M$ is the contract text X wants to sign, and $tid \in TIDs$ is the common unique[6] transaction identifier which is used to distinguish in- and outputs as well as messages from different protocol runs and which signals that both inputs belong together. At the end, each of A and B returns a local output, which can take the following values: $(\texttt{signed}, C, tid)$ containing a contract text C or $(\texttt{rejected}, tid)$.

Verification (Protocol "`show`*"):* This is the contract verification protocol between the verifier V and one of the signatories A or B[7]. The signatory, say A, obtains a local input $(\texttt{show}, tid)$. A does not make a local output. The verifier V outputs either $(\texttt{signed}, C, tid)$ or $(\texttt{rejected}, tid)$.

□

Intuitively, an output $(\texttt{signed}, C, tid)$ of the "`sign`"-protocol means that the user can now safely act upon the assumption that a contract "C" has been signed, i.e., that a subsequent verification will succeed. If the protocol outputs $(\texttt{rejected}, tid)$, the user can safely assume that no contract was signed, i.e., the other signatory will not be able to pass verification.

We now define the security requirements for contract signing depending on the underlying network. Since on asynchronous networks, nobody can decide whether the input will eventually arrive or not, termination cannot be guaranteed in general. Therefore, we allow the user to "switch" the model manually: After a local input $(\texttt{wakeup}, tid)$, the protocol stops waiting for pending messages and is required to terminate with a correct output. In practice, `wakeup` can be produced by a local time-out or by an interaction with the user.

Definition 3 (Fair Contract Signing)
A contract signing scheme (Def. 2) is called fair if it fulfills the following requirements:

Correct Execution: Consider an execution of "`sign`" by two correct signatories A and B on input $(\texttt{sign}, C_A, tid)$ to A and $(\texttt{sign}, C_B, tid)$ to B with a unique and fresh $tid \in TIDs$ and $C_A, C_B \in M$. Then, the "`sign`"-protocol outputs $(\texttt{signed}, C_A, tid)$ iff $C_A = C_B$ or else $(\texttt{rejected}, tid)$ to both signatories if none inputs `wakeup`.

Unforgeability of Contracts: If a correct signatory, say A, did not receive an input $(\texttt{sign}, C, tid)$ so far, a correct verifier V will not output $(\texttt{signed}, C, tid)$.

Verifiability of Valid Contracts: If a correct signatory, say A, output $(\texttt{signed}, C, tid)$ and later executes "`show`" on input $(\texttt{show}, tid)$ then any correct verifier V will output $(\texttt{signed}, C, tid)$.

No Surprises with Invalid Contracts: If a correct signatory, say A, output $(\texttt{rejected}, tid)$ then no correct verifier will output $(\texttt{signed}, C, tid)$ for any C.

[6] The parties must have agreed upon this before starting a contract signing protocol. A common method to guarantee uniqueness is to use a pair of two locally unique numbers as the global transaction identifier. In practice, a separate agreement on a *tid* may not be necessary since contract signing will be part of a larger commerce protocol.

[7] Here, we restrict our model for the moment to two-party verification: Three-party verification between A or B, V, and a third party is considered in Section 7.

Termination on Synchronous Network: A correct signatory, say A, will either output $(\texttt{rejected}, tid)$ or $(\texttt{signed}, C_A, tid)$ after a fixed number of rounds.

Termination on Asynchronous Network: On input $(\texttt{wakeup}, tid)$, a correct signatory, say A, will either output $(\texttt{signed}, C_A, tid)$ or $(\texttt{rejected}, tid)$ after a fixed time.

□

The requirement on "Verifiability of Valid Contracts" models that a contract that was *ever* declared `signed` by a correct signatory cannot be invalidated again. This means that one can safely buy a new house with the money if the protocol output `signed`. Similarly, the requirement on "No Surprises with Invalid Contracts" models that a contract which was ever declared `rejected` cannot be declared `signed` afterwards. This means that one can safely look for another buyer for the old house if one thinks no contract was signed. The "Unforgeability" requirement models that no valid contract can be produced without participation of a correct signatory.

Since it was mentioned in the literature [Even 83], we now define a weaker notion of contract signing schemes which enables more efficient but less practical protocols which we will later compare with our more restricted notion of a contract signing scheme:

Definition 4 (Contract Signing Scheme with Three-Party Verification)
A *contract signing scheme with three-party verification* is a contract signing scheme (Definition 2) where the definition of the verification protocol is changed as follows:

Verification (Protocol "`show`*"):* This is the contract verification protocol between the verifier V, the third party T, and one of the signatories A or B. The signatory, say A, obtains a local input $(\texttt{show}, tid)$. A does not make a local output. The verifier V outputs either $(\texttt{signed}, C, tid)$ or $(\texttt{rejected}, tid)$.

□

This changed model enables the third party to participate in the verification protocol which enables us to revoke sent parts of a contract: The basic mechanism used for revoking a signature works as follows: Once a party signed its part of a contract for a given contract signing execution identified by tid, the third party may tag this execution as being revoked if one of the signatories misbehaves. If somebody presents a contract containing this tid to a verifier, the verifier asks the third party if this execution has been revoked or not. If the contract is correct and the execution has not been revoked, the verifier decides on `signed` and else on `rejected`.

Recall, however, that we maintain the "Termination"-Requirement, i.e., for honest parties, revocations must not make any difference after the "`sign`"–protocol terminated.

2.3 Optimistic Contract Signing

To guarantee fairness, "optimistic contract signing" includes an additional third party T which is assumed to be correct in order to guarantee fairness. We try to limit the involvement of this third party by distinguishing two phases of the "`sign`"-protocol:

The *optimistic phase* tries to produce a contract without contacting the third party. Since a contract requires inputs from both signatories, this protocol may not terminate on asynchronous networks if a peer is not correct.

The *error recovery phase* is started if an exception, such as a wrong or missing message or the input of `wakeup`, occurs. In this phase, the third party is asked to guarantee a fair decision in a limited time. In our schemes, this phase is implemented by a sub-protocol called "`resolve`".

Definition 5 (Optimistic Contract Signing)
A fair contract signing scheme (Def. 3) is called *optimistic* iff an additional correct player T participates in the "`sign`"-protocol so that one of the following requirements is fulfilled:

Optimistic on Synchronous Network: If both signatories are correct, the third party does not send or receive messages in the "`sign`"-protocol.

Optimistic on Asynchronous Network: If both signatories are correct and do not input $(\texttt{wakeup}, tid)$, the third party does not send or receive messages in the "`sign`"-protocol.

□

The requirement that `wakeup` must not be input on asynchronous networks models that a user has to be patient in order to enable the protocol to terminate without involving the third party: If a user inputs `wakeup` immediately, the protocol may always involve the third party.

A weaker notion of optimistic contract signing requires optimistic execution only if the signatories input identical contract texts:

Definition 6 (Optimistic on Agreement)
A fair contract signing scheme (Def. 3) is called *optimistic on agreement* iff an additional correct player T participates in the "`sign`"-protocol so that one of the following requirements is fulfilled:

Optimistic on Synchronous Network: If both signatories are correct and both input $(\texttt{sign}, C, tid)$ with a fresh and unique tid and a $C \in M$, the third party does not send or receive messages in the "`sign`"-protocol.

Optimistic on Asynchronous Network: If both signatories are correct and both input $(\texttt{sign}, C, tid)$ with a fresh and unique tid and a $C \in M$ and do not input $(\texttt{wakeup}, tid)$, the third party does not send or receive messages in the "`sign`"-protocol.

□

2.4 Notations and Assumptions

All our schemes are based on a secure digital signature scheme [GoMR 88, RSA 78] where $\mathrm{sig}_{\mathsf{X}}(m)$ denotes X's signature under message m. Each party can sign messages, and can verify messages signed by others. All our protocols and definitions are formulated as if these digital signatures would provide error-free authentication. Furthermore, we assume tacitly that sequence numbers, names of participants, and the tid are included into all signed messages and that the signatures contained in messages are verified upon receipt. Corrupted or unexpected messages are just ignored.

All schemes are described by means of the message flows in the optimistic case, detailed descriptions of all protocols as well as figures of the states and transitions of the machines for signatories and third party. The verifier is not depicted as a state-machine but only described in the text: It is state-less and has only one state. In order to avoid unnecessarily complicated descriptions, we assume that the parties involved are a priori fixed. For synchronous protocols, we furthermore assume that all parties agree on the starting round of a protocol which is included in all messages.

In our figures, Ⓐ $\overset{a/b}{\rightarrow}$ Ⓑ depicts that a machine is in state A and receives message called a. It sends message called b and changes to state B. Dashed arrows denote exception handling by means of executing "`resolve`". If the message name is bold, the message is exchanged with the third party. Subscripts in message names usually denote the time at which they are sent (e.g., m_3 would be a message from round 3). Bold states are final states. If a message a is *not* received on a synchronous network, this is modeled by receiving the message $\neg a$.

3 A Message-Optimal Synchronous Scheme

Our message-optimal optimistic scheme[8] for synchronous networks requires three messages in the optimistic case using a state-less third party. Its optimistic behavior is depicted in Figure 1. The individual machines of the players are depicted in Figures 2, 4, and 3.

Scheme 1 (Message-Optimal Synchronous)
This scheme consists of the triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "`sign`*"; Figure 1):* On input $(\texttt{sign}, C_A, tid)$, the signatory A initiates the protocol by sending the signed[9] message $m_1 := \mathrm{sig}_{\mathsf{A}}(C_A)$ with contract C_A to the responding signatory B. B receives the input $(\texttt{sign}, C_B, tid)$ and message m_1 and verifies whether the received contract text C_A is identical to the input contract text C_B. If not, the players disagree about the contract and B returns $(\texttt{rejected}, tid)$. Else, it signs the received message and sends it as $m_2 := \mathrm{sig}_{\mathsf{B}}(m_1)$ to A. Player A then signs the received message again, sends it as $m_3 := \mathrm{sig}_{\mathsf{A}}(m_2)$ back and outputs $(\texttt{signed}, C, tid)$. On receipt of message m_3, B outputs $(\texttt{signed}, C, tid)$ as well. After a successful execution of this optimistic protocol, A and B store m_3 under the tid for later use in a verification protocol.

If A does not receive message m_2 it waits until Round 5, and, if m_5 was not received, it outputs $(\texttt{rejected}, tid)$. If B did not receive message m_3, it may be that A nevertheless was able to compute a valid contract m_3 after receiving m_2. Therefore it starts the "`resolve`"-protocol to invoke the third party to guarantee fairness.

Recovery from Exceptions (Sub-Protocol "`resolve`*"):* B sends a message $m_4 := \mathrm{sig}_{\mathsf{B}}(m_2)$ containing m_1 and m_2 to the third party T. The third party checks whether both players agreed and then forwards m_2 in $m_5 := m_2$ to A which might still wait for it. This guarantees that A receives a valid contract $m_3 := \mathrm{sig}_{\mathsf{A}}(m_2)$ and outputs $(\texttt{signed}, C_A, tid)$. Furthermore T sends an affidavit on m_2 in $m_5' := \mathrm{sig}_{\mathsf{T}}(m_2)$ to B and B outputs $(\texttt{signed}, C, tid)$. After the "`resolve`"-protocol, A keeps m_3 and B keeps m_5' to be used in later verification protocol executions.

Verification of a Contract (Protocol "`show`*"):* On input $(\texttt{show}, tid)$, a signatory looks up m_3 or m_5' and sends it to the verifier. The verifier verifies it and outputs $(\texttt{signed}, C, tid)$ if this succeeds and $(\texttt{rejected}, tid)$ else.

□

Lemma 1 (Security of Scheme 1)
Scheme 1 is a synchronous fair optimistic contract signing scheme. ◇

Proof of Lemma 1: The scheme adheres to Definition 2 by construction. We now show that each of the requirements described in Definitions 3 and 5 are fulfilled:

Correct Execution: If both correct players A and B input $(\texttt{sign}, C, tid)$ with identical tid and C, then both receive a valid contract m_3 and output $(\texttt{signed}, C, tid)$. If the contracts or tid's differ, B outputs $(\texttt{rejected}, tid_B)$ after receiving m_1 and A outputs $(\texttt{rejected}, tid_A)$ after not receiving m_5 in Round 5.

Unforgeability of Contracts: In order to convince a correct verifier V for a given tid, one needs correct messages m_3 or m_5' for this tid. Since m_3 as well as m_5' contain signatures from both participants, a correct signatory input $(\texttt{sign}, C, tid)$.

[8] The message flows are similar to the optimistic protocol in [Mica 97] which provides certified mail instead of contract signing.

[9] Note that in our protocols, the contract and the contents of most messages are fixed after the first message sent by a signatory. Therefore, each player can save signatures by including commitments to random authenticators r_i into the initial message which are then released instead of signing messages [AsSW 97].

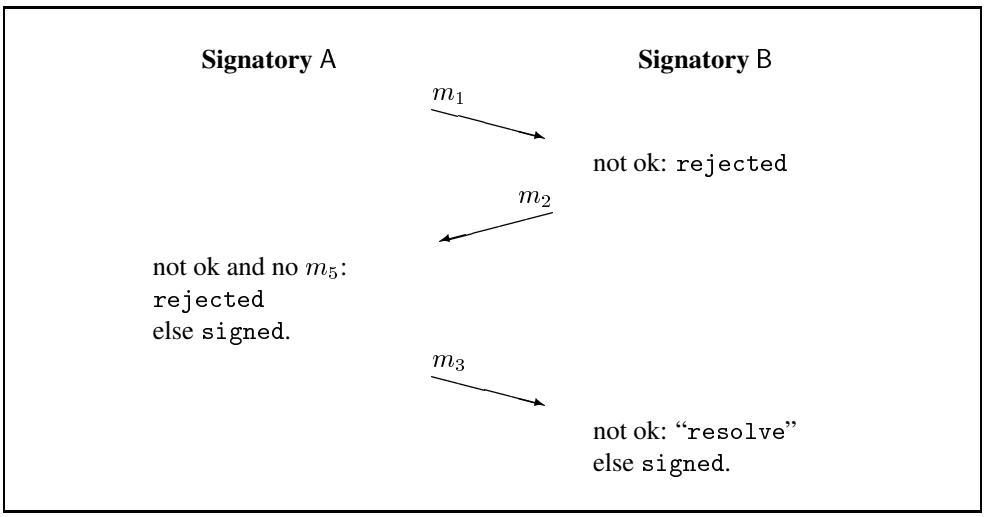

Figure 1: Optimistic Behavior of the Message-Optimal Synchronous Scheme 1.

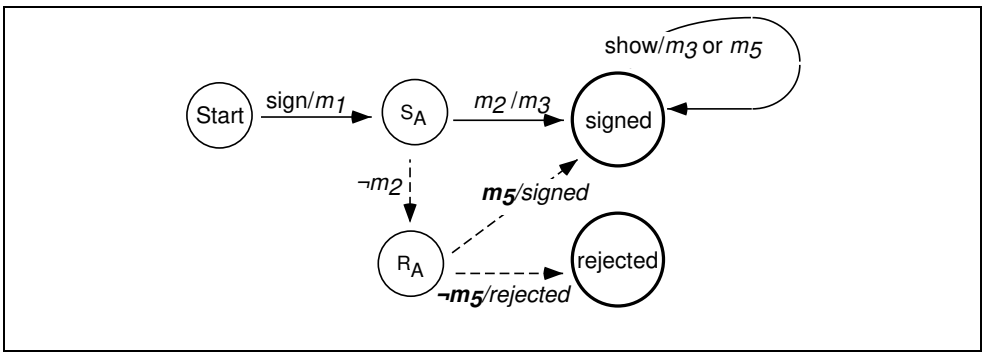

Figure 2: Signatory A of Scheme 1.

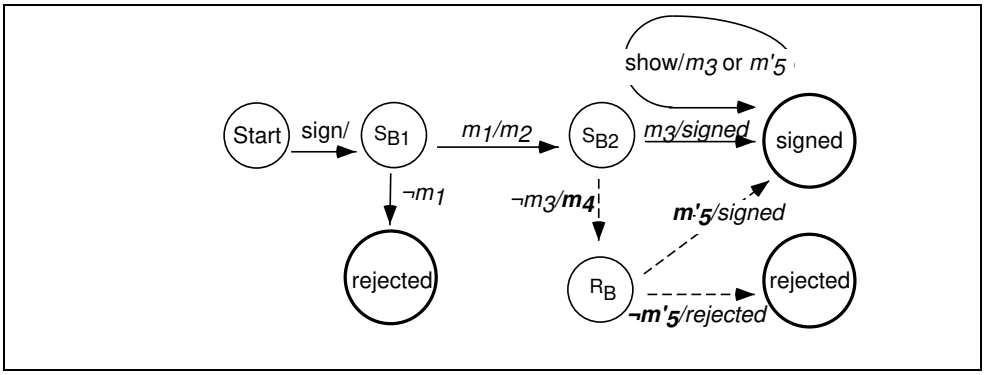

Figure 3: Signatory B of Scheme 1.

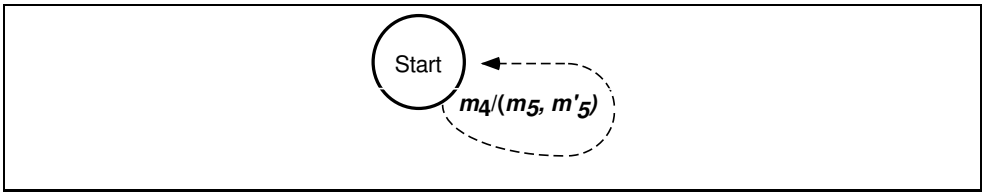

Figure 4: Third Party T of Scheme 1.

Verifiability of Valid Contracts: If A outputs (`signed`, C, tid) then it received m_2 (or m_5 containing m_2) which will be accepted by the verifier as a correct contract m_3 after being signed by A. B outputs (`signed`, C, tid) only if it received m_3 or m_5' which are accepted, too.

No Surprises with Invalid Contracts: Let us first assume that a correct signatory A returned `rejected` on input (`sign`, C, tid) whereas B is able to convince the verifier. This requires that B knows m_3 or m_5' for the given tid and C. Since A returned `rejected`, it did not receive m_2 until Round 5 and it did not send m_3. Therefore, only m_5' could lead to successful verification. However, if the third party was correct, it will not accept recovery requests from B after Round 4. Furthermore, in Round 4, no recovery was started since A did not receive m_5 in Round 5. Thus B did not receive m_5' in Round 5. Now let us assume secondly that A invokes "`resolve`". This would not cause problems since in this case, A needs m_2. Thus the contract will be validated anyway: Either B receives m_3 or it will start "`resolve`", too.

Termination on Synchronous Network: The scheme requires at most 5 rounds (3 in "`sign`" and 2 in "`resolve`") to terminate.

Optimistic on Synchronous Network: If two correct signatories input (`sign`, C, tid), signatory A outputs (`signed`, C, tid) after round 2 whereas player B outputs (`signed`, C, tid) after round 3. If they disagree, i.e., input different contracts, A outputs (`rejected`, tid) after Round 5 and B after Round 1 without contacting the third party by starting "`resolve`".

■

We now show that no optimistic contract signing scheme with only two messages exists. This proves that the number of messages of Scheme 1 is optimal. Furthermore, we show that it cannot be done with three messages in two rounds. Thus, the number of rounds of Scheme 1 is optimal, too, given the restriction to 3 messages.

Theorem 1 (Optimality of Scheme 1)
In the synchronous model with state-keeping third party, there exists no contract signing scheme which is optimistic on agreement with a "`sign`"–protocol with less than 3 messages in case of agreement and a protocol which needs 3 messages needs at least 3 rounds. ◇

Proof of Theorem 1: Let us assume that there exists an optimistic contract signing scheme which requires three messages in two rounds in case of agreement. In the optimistic phase, one player, say B, sends two messages m_{1B} in Round 1 and m_{2B} in Round 2.

Let us first assume that A sends its single message m_A in Round 1. Since two correct players who input identical contracts $C_A = C_B$ must not contact the third party this means that the single message m_A from A needs to be sufficient to enable B to convince the verifier. Now assume that an incorrect B receives the valid contract m_A but sends nothing. Then A must be able to obtain a valid contract since the contract m_A sent to B cannot be invalidated again (verification is a protocol between B and the state-less verifier V only). Therefore, player A needs to be able to start recovery without any input from B and a dishonest B must not be able to prevent this. This would enable A to forge a contract.

If we now assume, on the other hand, that A sends m_A in Round 2, m_A and m_{2B} must be valid contracts, i.e., sufficient for "`show`". If A now omits sending m_A, it will end up with a valid contract. Therefore B must be enabled to run "`resolve`" if A did not send its only message. The resulting recovery without any message from A, however, again contradicts the unforgeability requirement. Thus no protocol with 3 messages in 2 rounds exist.

If a two-message scheme exists, adding an empty message would produce a 3 message scheme in 2 rounds which does not exist. ■

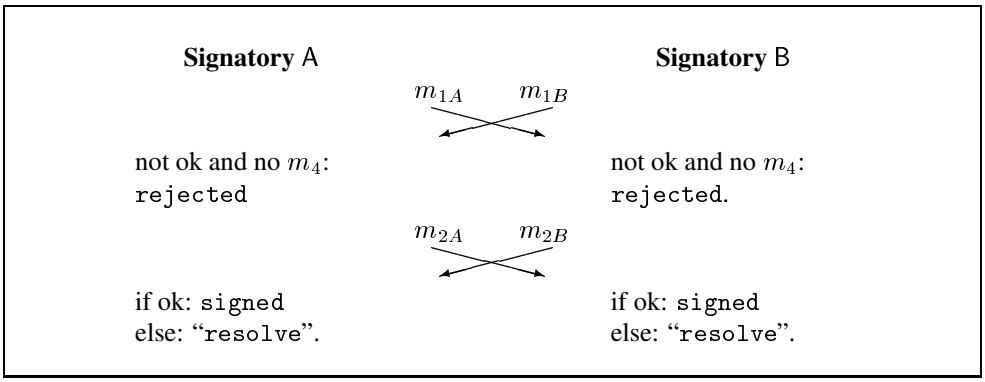

Figure 5: Optimistic Behavior of the Round-Optimal Synchronous Scheme 2.

4 A Round-Optimal Synchronous Scheme

We now describe the round-optimal Scheme 2 for synchronous networks and prove its security in Lemma 2. It requires only two rounds but four messages. Since any three-message "sign"-protocol needs at least three rounds (Theorem 1), there exists no one-round protocol at all and no 2-round protocol with only three messages. So the scheme described is optimal with respect to rounds and given the limitation to two rounds also with respect to the number of messages. The optimistic behavior of the scheme is depicted in Figure 5. The players are depicted in Figures 6 and 7.

Scheme 2 (Round-Optimal Synchronous)
This scheme consists of the triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "sign"; Figure 5): On input (sign, C_A, tid) a signatory, say A, sends message $m_{1A} := \mathrm{sig}_{\mathsf{A}}(C_A)$ in the first round. If it does not receive a message m_{1B} with $C_A = C_B$, it waits for recovery message m_4 and outputs (rejected, tid) if m_4 is not received in Round 4. If a message m_{1B} with $C_A = C_B$ is received, is sends $m_{2A} := \mathrm{sig}_{\mathsf{A}}(m_{1A}, m_{1B})$ in the second round and waits for m_{2B}. If m_{2B} with a correct contract text $C_A = C_B$ is received, it outputs (signed C_A, tid). Else, it starts "resolve".

Recovery from Exceptions (Sub-Protocol "resolve"): A signatory, say A, sends $m_{3A} := m_{2A}$ to the third party which verifies its consistency and signs an affidavit. This affidavit is sent as $m_4 := \mathrm{sig}_{\mathsf{T}}(m_{2A})$ to both parties. If the parties receive an affidavit in Round 4, they output (signed, C, tid). Else, they output (rejected, tid).

Verification of a Contract (Protocol "show"): On input (show, tid), a signatory, say A, looks up (m_{2A}, m_{2B}) or m_4 and sends it to the verifier V. The verifier checks that the signatures are correct. If these checks fail, it outputs (rejected, tid) and else (signed, C, tid).

□

Lemma 2 (Security of Scheme 2)
Scheme 2 is a synchronous fair optimistic contract signing scheme. ◇

Proof of Lemma 2: The scheme adheres to Definition 2 by construction. We now show that it fulfills the requirements stated in Definitions 3 and 5:

Correct Execution: If both players behave correctly and input identical tid's and contracts, each signatory, say A, receives m_{1B} and m_{2B}. Thus, the protocol outputs (signed, C, tid) on both machines. If the signatories disagree, both will receive inconsistent messages in Round 1 and will wait for recovery until Round 4. Since no recovery message m_4 will be received, they will output (rejected, tid).

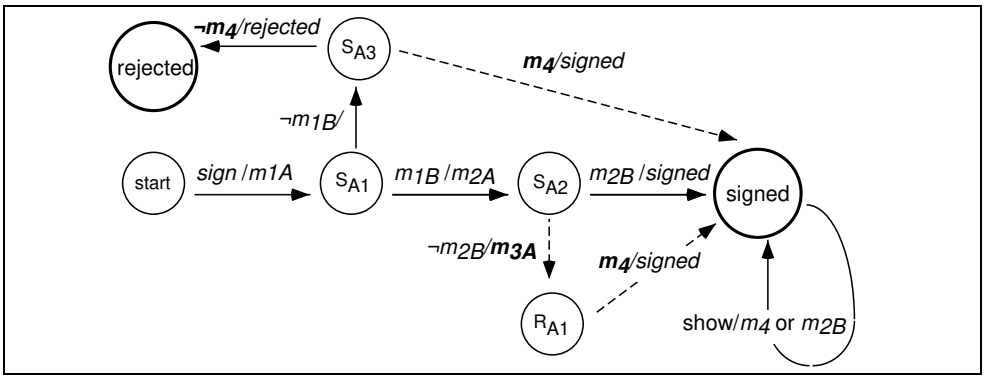

Figure 6: Signatory, e.g., A, of Scheme 2.

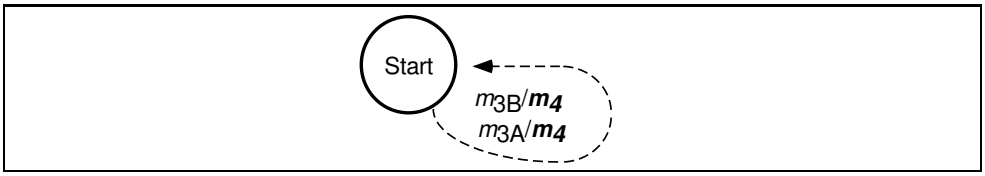

Figure 7: Third Party T of Scheme 2.

Unforgeability of Contracts: In order to convince a correct verifier, a signatory, say A, needs (m_{2A}, m_{2B}) or m_4. Since (m_{2A}, m_{2B}) as well as m_4 contain signatures from both signatories, a correct signatory input $(\texttt{sign}, C, tid)$.

Verifiability of Valid Contracts: A signatory, say A, only outputs $(\texttt{signed}, C, tid)$ after receiving m_4 or after sending m_{2A} and receiving m_{2B}. Thus, they are able to convince the verifier.

No Surprises with Invalid Contracts: If a signatory, say A, decides `rejected`, this signatory did not start "`resolve`" and did not receive m_4 in Round 4 which means that B also did not receive m_4. In order to convince a verifier, B therefore needs m_{2A}. However, since A output `rejected`, it did not send m_{2A}.

Termination on Synchronous Network: At most 4 rounds are required for termination.

Optimistic on Synchronous Network: If two correct signatories input $(\texttt{sign}, C, tid)$, they output $(\texttt{signed}, C, tid)$ after round 2 without contacting the third party. If they disagree, they output $(\texttt{rejected}, tid)$ after round 4 without contacting the third party.

■

5 A Time-Optimal Asynchronous Scheme

We now describe a new time-optimal asynchronous contract signing scheme. It terminates in time 3 and requires six messages in the optimistic case. In Theorem 3 we prove that this is time-optimal. Its optimistic behavior is sketched in Figure 8, the machines are depicted in Figures 9 and 10. Note that the third party is state-keeping: Once a contract is accepted (i.e., m_5' or m_5'' was sent), the third party enters its `signed` state which disables aborting the protocol. A state-less third party would be more convenient, but we prove in Theorem 4 that this is not possible.

A message-optimal scheme has been proposed in [AsSW3 97]. It describes an asynchronous scheme which requires four consecutive messages and time four. This is message-optimal in the optimistic case since, as we will prove, there is no asynchronous optimistic contract signing scheme with only three messages (Theorem 2).

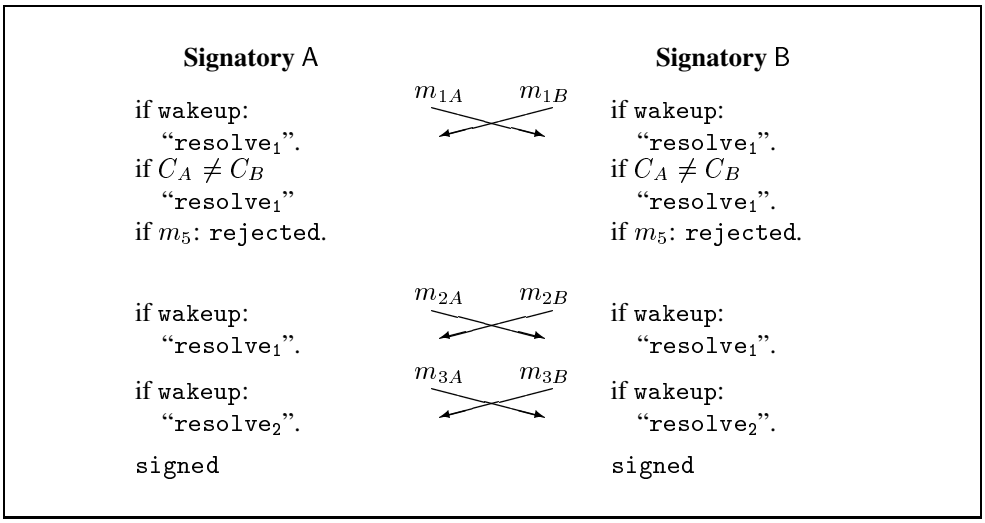

Figure 8: Optimistic Behavior of the Time-Optimal Asynchronous Scheme 3.

Scheme 3 (Time-Optimal Asynchronous)
This scheme consists of the triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "`sign`*"; Figure 8):* On input $(\texttt{sign}, C_A, tid)$ the signatory, say A, sends its signed contract in message $m_{1A} := \mathrm{sig}_{\mathsf{A}}(C_A)$. If A receives m_{1B} with an identical contract, it sends $m_{2A} := \mathrm{sig}_{\mathsf{A}}(m_{1A}, m_{1B})$. If a message m_{2B} from B is received, A sends $m_{3A} := \mathrm{sig}_{\mathsf{A}}(m_{2A}, m_{2B})$. After receiving m_{3B}, the signatory outputs $(\texttt{signed}, C, tid)$. If m_{2B} is received before m_{1B} since the messages have been reordered by the asynchronous network, both m_{2A} and m_{3A} are sent. If m_{3B} is received before m_{2B}, m_{3A} is sent and $(\texttt{signed}, C, tid)$ is output. If a m_{1B} with a different contract is received before m_{2B} or if $(\texttt{wakeup}, tid)$ occurs before m_{2A} has been sent, "$\texttt{resolve}_1$" is started by sending $m_{4A} := \mathrm{sig}_{\mathsf{A}}(m_{1A})$, if it occurs after m_{2A} has been sent but before m_{3A}, "$\texttt{resolve}_1$" is started by sending $m'_{4A} := \mathrm{sig}_{\mathsf{A}}(m_{2A})$, else "$\texttt{resolve}_2$" is started by sending $m''_{4A} := \mathrm{sig}_{\mathsf{A}}(m_{3A})$. Messages m_{2B} or m_{3B} from a cheating player B containing different contracts $C_A \neq C_B$ are ignored.

Recovery from Exceptions (Sub-Protocol "$\texttt{resolve}_1$*"):* This protocol is used in a situation where the status of a contract may not be clear. If the signatory sends m_{4A} to abort the protocol, the third party either resends a previously sent decision m_5, m'_5 or m''_5 or else an abort acknowledgment $m_5 := \mathrm{sig}_{\mathsf{T}}(m_{4A})$ and changes to the `aborted`-state for the aborting signatory. If the signatory sends m'_4, the third party either resends a previous decision m_5, m'_5, or m''_5 or else signs an affidavit $m'_5 := \mathrm{sig}_{\mathsf{T}}(m'_4)$. After receiving m_5, the signatory outputs $(\texttt{rejected}, tid)$. After receiving m'_5 or m''_5, the signatory outputs $(\texttt{signed}, C, tid)$.

Recovery from Exceptions (Sub-Protocol "$\texttt{resolve}_2$*"):* This recovery sub-protocol is used to complete the contract if it is clear that the signatories agreed on the contract text. One signatory, say A, sends its message m''_{4A} to the third party. The third party then either resends a previous decision m'_5 or m''_5 or else produces an affidavit and sends it as $m''_5 := \mathrm{sig}_{\mathsf{T}}(m''_{4A})$ to A who outputs $(\texttt{signed}, C, tid)$. This recovery by A overrides the effects of a previous abort message m_{4B} sent by an incorrect player B.

Verification of a Contract (Protocol "`show`*"):* After the input $(\texttt{show}, tid)$, a signatory, say A, looks up (m_{3A}, m_{3B}), m'_5, or m''_5 and sends it to the verifier V. The verifier verifies the messages. If these checks fail, it outputs $(\texttt{rejected}, tid)$ and else $(\texttt{signed}, C, tid)$.

□

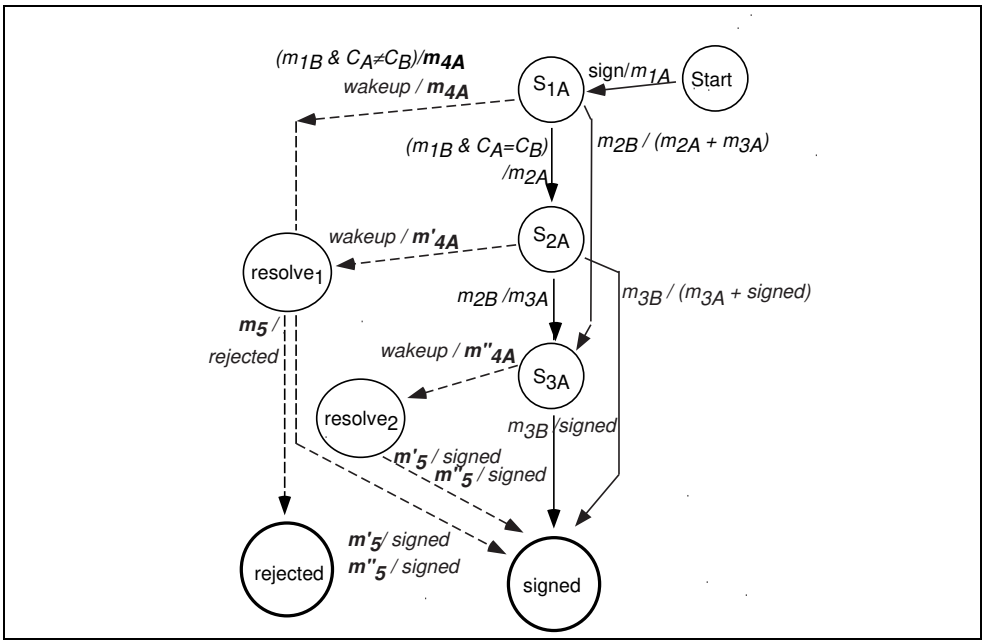

Figure 9: Signatory, e.g., A, of Scheme 3 (states S_{2A} or S_{3A} may be bypassed if messages are reordered).

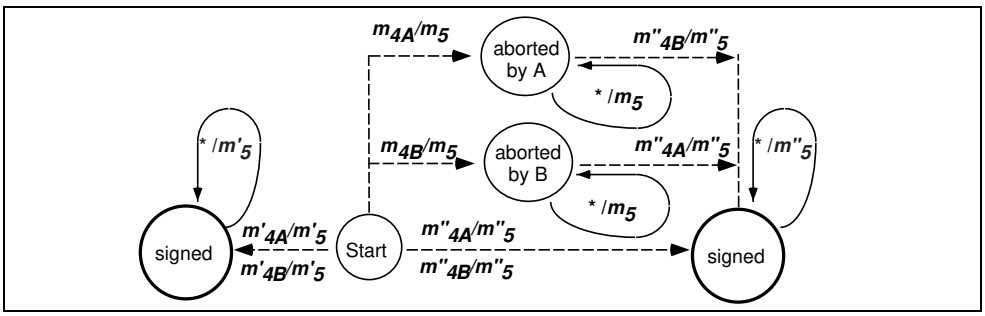

Figure 10: Third Party T of Scheme 3.

Lemma 3 (Security of Scheme 3)
Scheme 3 is an asynchronous fair contract signing scheme which is optimistic on agreement. ◇

Proof of Lemma 3: Scheme 3 adheres to Definition 2 by construction. We now show that it also fulfills the requirements stated in Definitions 3 and 6.

Correct Execution: If both signatories A and B start with identical inputs (`sign`, C_A, tid) and (`sign`, C_B, tid) and do not input `wakeup` then both will eventually receive all messages and will output (`signed`, C, tid). If they disagree, both will abort by sending m_4 and will finally output (`rejected`, tid).

Unforgeability of Contracts: Assume that a correct verifier outputs (`signed`, C, tid). This means that he received at least messages m_{1A}, m_{1B} (maybe included in m'_5 or m''_5) containing identical contracts which are signed by A and B, respectively. Thus, all correct parties have input (`sign`, C, tid) since otherwise they would not have sent m_{1A} and m_{1B} at all.

Verifiability of Valid Contracts: A signatory, say A only outputs (`signed`, C, tid) after receiving m_{3B} or m'_5 or m''_5 containing identical contracts in messages m_{1A} and m_{1B}. Thus, it is able to convince a verifier in all cases.

No Surprises with Invalid Contracts: Let us assume that (`rejected`, tid) was output by

a correct signatory, say A, after receiving m_5 and a correct verifier invoked by B outputs (`signed`, C, tid). Then either (m_{3A}, m_{3B}), m'_5 or m''_5 must be known by B. Let us first assume that (m_{3A}, m_{3B}) was shown to the verifier then A sent both m_{3A} and m_{4A} or m'_{4A}, i.e., A was incorrect. Let us now assume that m'_5 was shown to the verifier then T sent both m_5 and m'_5, i.e., the third party was incorrect. Let us finally assume that m''_5 was shown to the verifier. Since m_5 as well as m''_5 were produced by the third party, the machine T was in one of the aborted states and thus A must have sent either m_{4A} or m''_{4A}. Since A received m_5, it did not send m''_{4A}. Together this implies that A sent m_{4A}. This contradicts the assumption that m''_5 was shown to the verifier a correct A which sends m_{4A} does not send m_{2A} which is part of m''_5.

Termination on Asynchronous Network: If the user inputs `wakeup`, a "`resolve`"-protocol is started. In this protocol, the other signatory is not involved anymore. Since the third party is assumed to be correct, it will answer. Thus, the "`resolve`"-protocol terminates with a definitive answer after time 2, i.e., a fixed time after the input of `wakeup`.

Optimistic on Agreement: If two correct signatories do not input `wakeup` and input identical contracts, they both receive the outputs (`signed`, C, tid) from the "`sign`"-protocol after time 3 without contacting the third party.

■

We now prove in Theorem 2 that asynchronous contract signing with only 3 messages is impossible. Then we prove the optimality of Scheme 3 in Theorem 3. Since this scheme still needs a state-keeping third party, we will show in Theorem 4 that one cannot do better, i.e., that recovery with a state-less third party is not possible in the asynchronous case.

Theorem 2 (Message-Optimality of Scheme in [AsSW3 97])
There exists no asynchronous optimistic contract signing scheme with a "`sign`"–protocol with less than four messages in case of agreement. ◇

In order to prove this theorem, we first show that recovery cannot involve both signatories in the asynchronous case:

Lemma 4 (Asynchronous Recovery is 2-Party)
The outcome of the recovery phase on asynchronous networks is determined only by inputs from the third party and the signatory starting it. ◇

Proof of Lemma 4: If the third party is invoked by a correct player, the recovery phase is required to terminate in order to guarantee termination of the "`sign`"-protocol. However, if the third party asked the other signatory, the third party cannot decide whether the message would eventually be answered or not. Thus, if this signatory is not correct, "`resolve`" would not terminate. ■

Proof of Theorem 2: Let us assume that A sends two messages, say m_1 and m_3, in the optimistic phase whereas B sends only one message, say m_2. Then (m_1, m_2, m_3) must be sufficient to convince the verifier. If A sends m_1 and m_3 without having received m_2, B can convince a verifier without sending m_2. Therefore, A is required to be able to recover to `signed` without contacting B (Lemma 4) which contradicts the unforgeability requirement. Thus, m_3 is sent after m_2 has been received. If we now assume that B sends m_2 before receiving m_1, A could convince a verifier without sending any message and B would be required to be able to recover to `signed` without contacting A (Lemma 4) which again contradicts the unforgeability requirement.

Therefore, the messages are sent in the order m_1, m_2, m_3 (similar to Scheme 1 depicted in Figure 1). Since the protocol is optimistic, at least (m_1, m_2) shown by A and (m_1, m_2, m_3) shown by B are sufficient to convince the verifier. Now consider the exceptions: Let us assume that T did not decide for this tid before. If B now does not receive m_3, the third party has to decide locally (Lemma 4) on `signed` since A may have obtained a valid contract (m_1, m_2). Thus B may obtain a valid contract from the third party even if A

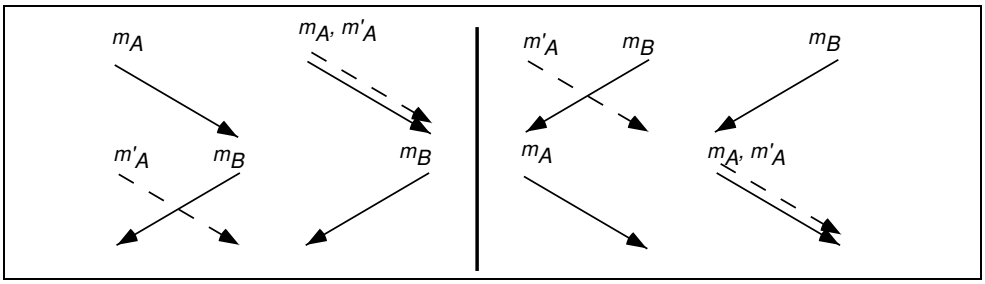

Figure 11: Proof of Theorem 3: Saving a Dashed Message by Shoving it Up or Down.

only sent m_1. Therefore, A must be able to start recovery with the third party after sending m_1, too. In this case, the third party is required to decide locally whether the contract is valid or not given only m_1 from A. For unforgeability for B, it has to decide on `rejected` based on m_1 only. If B now asks for recovery with m_1 and m_2, T has to decide locally. If it decides on `signed` since a dishonest A may have started recovery after receiving valid contract (m_1, m_2, m_3), the "no surprises" requirement for a correct A would not be fulfilled. If T decides on `rejected`, a correct player B may be surprised since A may have obtained a valid contract. ■

This enables us to prove the optimality of Scheme 3:

Theorem 3 (Optimality of Scheme 3)
There exists no asynchronous optimistic contract signing scheme with a "`sign`"–protocol in less than time 3 in case of agreement and a protocol in time 3 needs at least 6 messages. ◇

Proof of Theorem 3: If we assume that a 2-time 4-message optimistic "`sign`"-protocol exists, then this can be used to construct a 3-time 3-message protocol: Since the two-party signing protocol has 4 messages labeled with two subsequent times, two messages (m_{1A}, m_{1B}) are labeled with time 1 and two messages (m_{2A}, m_{2B}) are labeled with time 2 where each two messages labeled with the same time are independent from each other. Therefore, one player, say B, can send m_{1B} together with m_{2B} and m_{2A} can be sent after receiving these two messages. The result is a three-message protocol with the messages $m'_{1A} := m_{1A}$, $m'_{2B} := (m_{1B}, m_{2B})$, and $m'_{3A} := m_{2A}$ which does not exist according to Theorem 2.

If we assume that a 5-message protocol in time 3 exists, we can construct an equivalent protocol with 3-messages in time 3 by shoving a message up or down (see Figure 11): If 5 messages are sent in time 3, there exists a time t for which only one message m_A sent by one signatory, say A, exists. Furthermore, two messages m'_A and m_B are labeled with time t' which is either $t+1$ or $t-1$. If two messages are labeled with time $t+1$ then the messages m_A and m'_A can be sent together at time t. This is possible since A does not receive a message at time t which guarantees that the contents of m'_A have already been fixed when m_A was sent. For B, receiving m'_A earlier must not make a difference since the network may have reordered the messages anyhow. If, on the other hand, two messages m'_A and m_B are labeled with time $t-1$ then the messages m'_A and m_A can be sent together at time t. This is possible since B does not send a message at time t which implies that m'_A is not needed by B to compute a message. This construction enables to change two subsequent times with two and one messages into two subsequent times with one message each. Two applications of this construction result in the desired 3-message protocol in time 3 which contradicts Theorem 2. ■

Finally, we show that the state-keeping third party in Scheme 3 cannot be avoided:

Theorem 4 (Asynchronous T Keeps State)
There is no asynchronous contract signing scheme with state-less third party which is optimistic on agreement. ◇

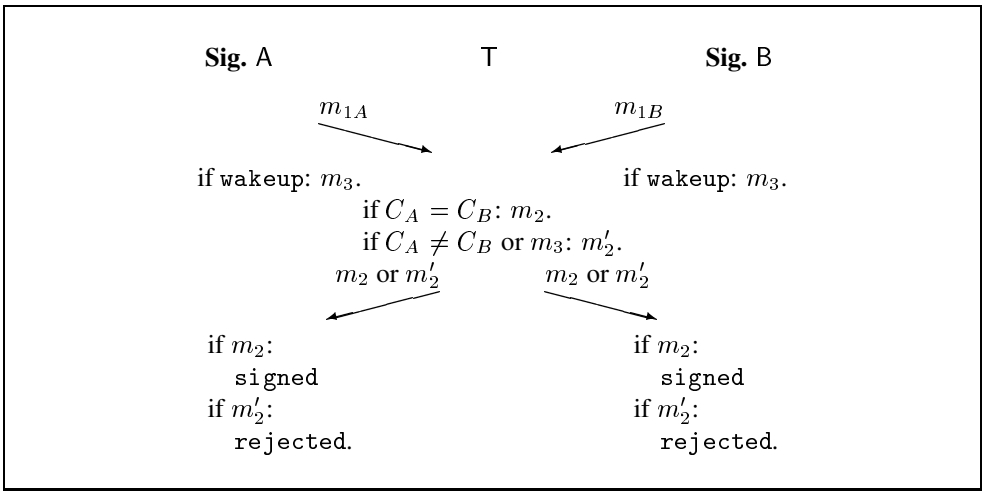

Figure 12: Behavior of the Optimal Asynchronous Scheme 4 with In-Line Third Party.

Proof of Theorem 4: Assume there is an asynchronous optimistic contract signing scheme. Then by means of the construction in the proof of Theorem 3, there is an equivalent "sign"-protocol which has only messages, say $m_1, \ldots, m_n$, in a row where A sends m_1 and m_n (if not, prepending an empty message helps). Furthermore, we assume that in "resolve", the third party gets all messages the invoker has sent or received so far, i.e., a prefix $(m_1, \ldots, m_k)$ of $(m_1, \ldots, m_n)$. Since we are in an asynchronous model, the third party's decision cannot depend on the non-invoking signatory (Lemma 4). Since the third party is assumed to be state-less, the decision is actually a set of functions $TP()$ on $(m_1, \ldots, m_k)$ to $\{\texttt{signed}, \texttt{rejected}\}$ for each k for which a request is allowed.

Consider a run with correct A and B where both input identical contracts and B inputs wakeup before the last message m_n from A has been received. Since A may have received a valid contract, the third party must decide $TP(m_1, \ldots, m_k) := \texttt{signed}$ for $k = n-1$ to fulfill the "No Surprises"-requirement.

Now assume that $TP(m_1, \ldots, m_k) = \texttt{signed}$ for some $k \geq 2$. If we now consider the case that the other player gets a wakeup after sending m_{k-1}, a recovery request must be allowed since the other player will eventually receive m_{k-1} which would enable it to recover to signed. For consistency reasons, we have $TP(m_1, \ldots, m_{k-1}) := TP(m_1, \ldots, m_k)$=signed. Thus, inductively we get $TP(m_1) = \texttt{signed}$ which contradicts the unforgeability requirement. ■

6 An Optimal Asynchronous Non-Optimistic Scheme

All protocols up to now were optimistic, i.e., the third party was only invoked in case of failures. We now prove the message and time optimality of an asynchronous version of a well-known synchronous fair exchange protocol based on a third party storing and forwarding the contract signatures.

This protocol needs four messages in time 2 and works on asynchronous networks. Its behavior is depicted in Figure 12. The machines for the individual players are depicted in Figures 13 and 14.

Scheme 4 (Time-Optimal Non-Optimistic Protocol)
This scheme consists of the triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "sign"; Figure 12): On input (sign, C, tid), each signatory, say A, sends a signed message $m_{1A} := \mathrm{sig}_{\mathsf{A}}(C_A)$ containing the contract text C_A

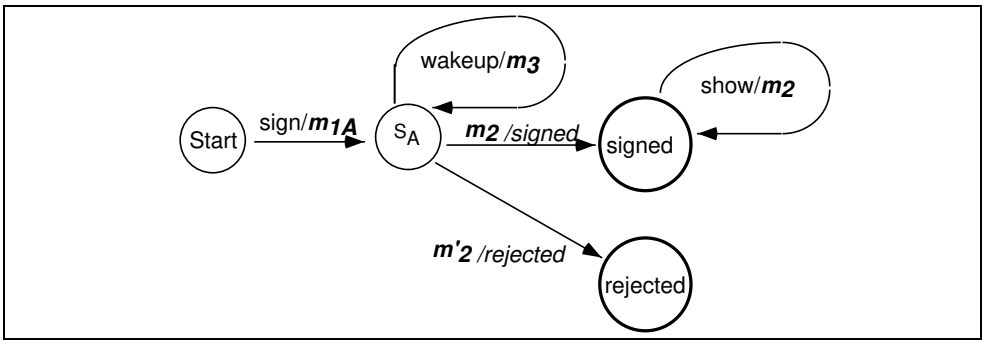

Figure 13: Signatories, e.g., A, of Scheme 4.

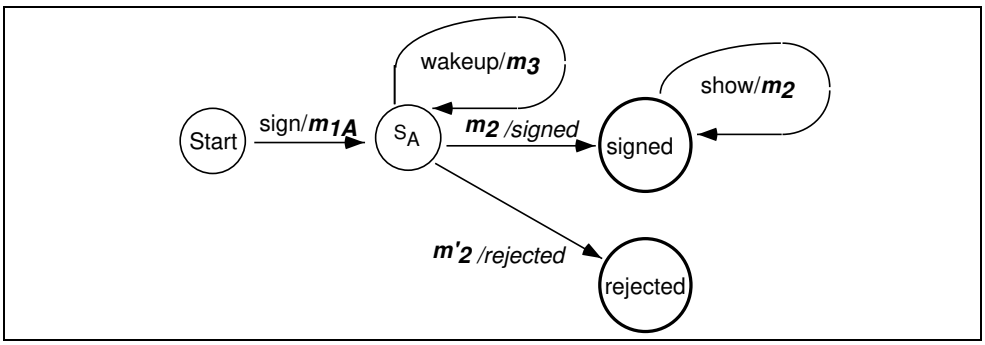

Figure 14: Third Party T of Scheme 4.

to the third party. If an input $(\texttt{wakeup}, tid)$ is made, a message $m_3 := \mathrm{sig}_{\mathsf{A}}(wakeup)$ is sent. The third party waits for m_{1A} and m_{1B} and verifies whether $C_A = C_B$ and $tid_A = tid_B$. If this is the case, the third party sends the message m_2:=$\mathrm{sig}_{\mathsf{T}}(m_{1A}, m_{1B})$ to both signatories. If the checks fail or `wakeup` is received before both messages have been received, the third party sends $m_2' := \mathrm{sig}_{\mathsf{T}}(\texttt{rejected})$.

Verification of a Contract (Protocol "`show`"): On input $(\texttt{show}, tid)$, signatory A looks up m_2 and sends it to the verifier. The verifier checks whether the message is valid and outputs $(\texttt{signed}, C, tid)$ if this succeeds and $(\texttt{rejected}, tid)$ else.

□

We now prove the security of this scheme:

Lemma 5 (Security of Scheme 4)
Scheme 4 is an asynchronous fair contract signing scheme if the machine T is correct. ◇

Proof of Lemma 5: The scheme adheres to Definition 2 by construction. We now show that each of the requirements described in Definition 3 are fulfilled:

Correct Execution: If both correct players A and B input $(\texttt{sign}, C, tid)$ with identical tid and C and do not input `wakeup`, then both receive a valid contract m_2. If the contracts or tid's differ, A and B output $(\texttt{rejected}, tid)$ after receiving m_2'.

Unforgeability of Contracts: In order to convince a correct verifier V for a given tid, one needs $m_2 := \mathrm{sig}_{\mathsf{T}}(m_{1A}, m_{1B})$ including this tid. A correct signatory, say A, will not send m_{1A} without the input $(\texttt{sign}, C, tid)$.

Verifiability of Valid Contracts: If A outputs $(\texttt{signed}, C, tid)$ then it received m_2 which will be accepted by the verifier as a correct contract.

No Surprises with Invalid Contracts: Let us assume that a correct signatory A returned `rejected` on input $(\texttt{sign}, C, tid)$ whereas B is able to convince the verifier. Then B received m_2 whereas A received m_2'. This implies that T sent m_2 and m_2' with the same tid which contradicts the assumption that the T is correct.

Termination on Asynchronous Network: If the user has input (`wakeup`, tid), the scheme requires at most time 2 to output `signed` or `rejected`.

■

We now prove the optimality of Scheme 4 in the case where the participants agree.

Theorem 5 (Optimality of Scheme 4)
There exists no asynchronous non-optimistic contract signing scheme with a "`sign`"–protocol with less than four messages or less than two rounds in case of agreement. ◇

Note that this theorem also proves that the message complexity (in the fault-less case) of the scheme in [AsSW3 97] cannot be improved upon by allowing the third party to participate in the "`sign`"-protocol in case of agreement.

Proof of Theorem 5: If we assume that a three-message protocol with third party exists, then the following prerequisites hold: If the third party sends messages without having received any message before, these messages are independent of the contract to be signed and can be omitted. If the third party does not send and receive any messages, the protocol is optimistic, and a three-message optimistic protocol where the third party does not participate in the verification does not exist (Theorem 2). If the third party only receives messages, these messages do not change the outputs of the signatories or the result of a subsequent verification because T does not participate in "`show`" and can be omitted. Therefore, the third party first receives some messages and then sends some. Any protocol where one of the signatories sends no messages contradicts the unforgeability requirement since the outcome will be independent of the contract input by this participant. If, on the other hand, a signatory does not receive any messages, the output of this signatory is independent of the contract input by the other signatory which contradicts our requirements, too.

Thus each of the three players send and receive one message each, and there are only three messages. Therefore, these messages are sent in a circle where T does not send the initial message and does not receive the last message, i.e., (A, B, T, A) or (A, T, B, A). In both cases, B must output `signed` after receiving its only message. By the verifiability requirement, B will then also produce an output `signed` at the verifier by showing this message. Moreover A may start recovery before anyone got a message from B, thus after A received a `wakeup`, A and T must be able to decide and recover locally (similar to Lemma 4). If this recovery leads to an output `signed` to A, it will contradict the unforgeability requirement for B, because T makes this decision without any input from B. If they recover to `rejected`, it contradicts the "No Surprises"- requirement for A because B has already received a valid contract.

To prove that no protocol in time 1 exists, we assume there were such a protocol. In this protocol, the messages from the third party cannot depend on the contract, whereas the messages to the third party will not change the outcome of a subsequent verification since we do not allow 3-party verification. Therefore, these messages can be omitted. The resulting protocol would be a two-message protocol, and we have already shown that those do not exist. ■

7 An Optimal Synchronous Scheme with Three-Party Verification

We will now give a precise description of the synchronous contract signing scheme sketched in [Even 83] and prove its correctness and its optimality. The scheme requires two messages in only one round. The underlying idea is that both send their contract and "no answer" means agreement. The optimistic behavior is depicted in Figure 15. The behavior of the signatories is depicted in Figure 16. The third party is depicted in Figure 17.

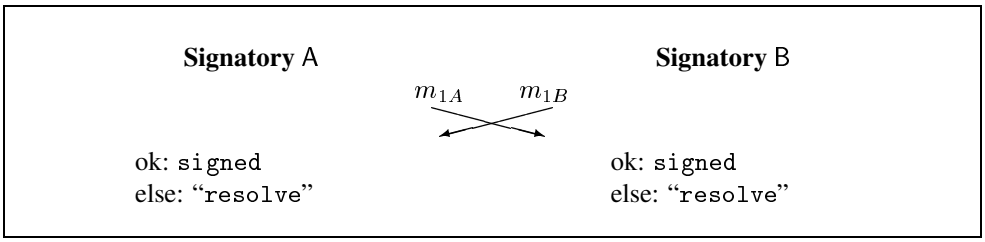

Figure 15: Optimistic Behavior of the Message-Optimal Synchronous Scheme 5 based on Signature Revocation.

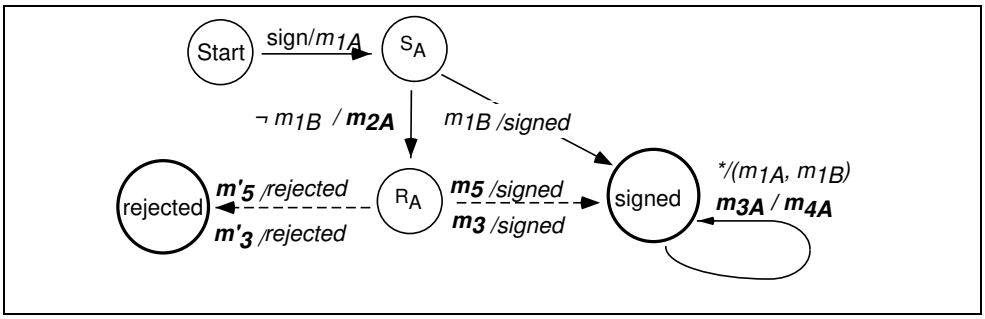

Figure 16: Signatory, e.g., A, of Scheme 5.

Scheme 5 (Scheme from [Even 83])
This scheme consists of the triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ and a third party T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "`sign`*"; Figure 15):* On input (`sign`, C_A, tid), each signatory, say A, sends its contract $m_{1A} := \mathrm{sig}_{\mathsf{A}}(C_A)$ to the other signatory in the first round. If a signatory receives m_{1B} with the same contract from the peer, it outputs (`signed`, C, tid). Else, it starts "`resolve`".

Recovery from Exceptions (Sub-Protocol "`resolve`*"):* One signatory, say player A, starts "`resolve`" in Round 2: A sends a message $m_{2A} := \mathrm{sig}_{\mathsf{A}}(m_{1A})$ to the third party.

If the third party receives messages m_{2A} and m_{2B} from both players in Round 2 it forwards them as $m_3 = \mathrm{sig}_{\mathsf{T}}(m_{1A}, m_{1B})$ or $m'_3 := \mathrm{sig}_{\mathsf{T}}(\texttt{revoke})$ to both players depending on whether the contained contracts were identical or not. Both players then output $(\texttt{signed}, C, tid)$ if they received m_3 and $(\texttt{rejected}, tid)$ if they received m'_3.

If the third party receives only one recovery request, say message m_{2A} from A, this message contains A's alleged part m'_{1A} of the contract. The third party forwards this message in Round 3 as $m_{3B} := \mathrm{sig}_{\mathsf{T}}(m_{2A})$ to B. Player B then resends the contract it has received in message $m_{4B} := \mathrm{sig}_{\mathsf{B}}(m_{1A}, m_{1B})$ to T who forwards it in $m_5 = \mathrm{sig}_{\mathsf{T}}(m_{1A}, m_{1B})$ to A who will output $(\texttt{signed}, C, tid)$. If a dishonest B does not answer, T revokes A's signature contained in m_{1A} by sending a revocation message $m'_5 := \mathrm{sig}_{\mathsf{T}}(\texttt{revoke})$ to A who will output $(\texttt{rejected}, tid)$.

Note that message m_{1A} contained in m_{4B} may be different from the message m'_{1A} contained in m_{2A}. In this case, an incorrect A sent a wrong recovery request m_{2A} and the contained m'_{1A} is just ignored.

Verification of a Contract (Protocol "`show`*"):* On input $(\texttt{show}, tid)$, a signatory, say A, looks up (m_{1A}, m_{1B}) and sends it to the verifier. The verifier forwards the contract to T. If T does not resend a revocation message m'_3 or m'_5 in the next round, the verifier outputs $(\texttt{signed}, C, tid)$ and $(\texttt{rejected}, tid)$ else.

□

Note that if both signatories as well as the third party participate in the verification (i.e.,

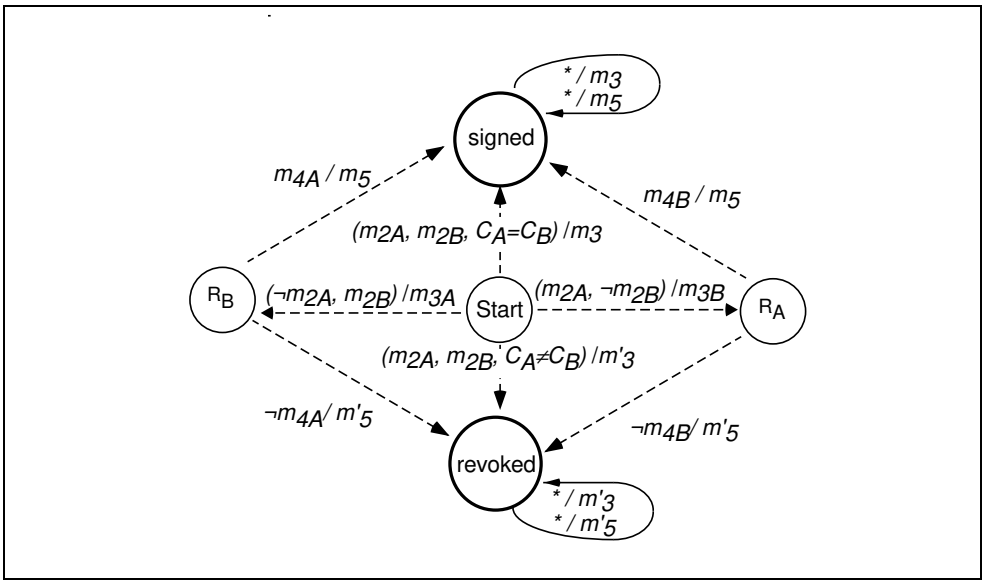

Figure 17: Third Party T of Scheme 5.

an even weaker model), one may make this scheme optimistic: The non-showing party may show a different signed message if the showing party cheats.

Lemma 6 (Security of Scheme 5)
Scheme 5 is a synchronous fair contract signing scheme with three-party verification which is optimistic on agreement. ◇

Proof of Lemma 6: The scheme adheres to the modified Definition 4 by construction. We now show that each of the requirements described in Definitions 3 and 6 are fulfilled:

Correct Execution: If both correct players A and B input (`sign`, C, tid) with identical tid and C, then both receive a valid contract and output (`signed`, C, tid). If both input different contracts, both will start resolve after Round 1 and the third party will send m'_3 to both signatories who will then output (`rejected`, tid).

Unforgeability of Contracts: In order to convince a correct verifier V for a given tid, one needs (m_{1A}, m_{1B}). A correct signatory, say A, will not send m_1 without the input (`sign`, C, tid).

Verifiability of Valid Contracts: If a correct player A outputs (`signed`, C, tid) then it has received m_{1B} and produced m_{1A} or else received m_3 or m_5 containing (m_{1A}, m_{1B}) with identical contract texts. This will be accepted by the verifier if this tid was not revoked. Now assume that a correct T sent a revocation token m'_3 or m'_5 for this tid. If the token is m'_3 (both parties started recovery), either T was incorrect since it sent m'_3 and m_3 or m_5 or else A was incorrect since it started `resolve` after a output `signed`. If the token was m'_5 (one party started recovery), either A or B did not answer. If a correct player A output `signed`, it would answer. If B did not answer, A would not have output `signed`.

No Surprises with Invalid Contracts: If an correct signatory A output `rejected` on input (`sign`, C, tid), it has received m'_3 or m'_5 which will be resent by the third party to the verifier during the verification protocol. Therefore, the verifier will decide on `rejected`, too.

Termination on Synchronous Network: The scheme requires at most 5 rounds (1 in "`sign`" and 4 in "`resolve`") to terminate.

Optimistic on Agreement: If two correct signatories input $(\texttt{sign}, C, tid)$, the protocol outputs $(\texttt{signed}, C, tid)$ to both participants after round 1 without contacting the third party.

■

Theorem 6 (Optimality of Scheme 5)
In the synchronous model with state-keeping third party, there exists no contract signing scheme with three-party verification and a "sign"–protocol with less than two messages in case of agreement. ◇

Proof of Theorem 6: If an optimistic 1-message protocol exists, one party does not send any message. Therefore, the third party and the sender of the single message are able to convince a verifier which contradicts the unforgeability requirement for the recipient of this single message. ■

8 A Optimal Asynchronous Scheme with Three-Party Verification

We now describe an asynchronous version of the optimistic Scheme 1. This scheme can only be made asynchronous by allowing three-party verifications. The individual machines of the players are depicted in Figures 19, 21, and 20.

Scheme 6 (Message-Optimal Synchronous)
This scheme consists of the triple $(\mathsf{A}, \mathsf{B}, \mathsf{V})$ and T of interactive probabilistic machines which are able to execute the protocols defined as follows:

Contract Signing (Protocol "sign"; Figure 18): On input $(\texttt{sign}, C_A, tid)$, the signatory A initiates the protocol by sending the signed message $m_1 := \mathrm{sig}_{\mathsf{A}}(C_A)$ with contract C_A to the responding signatory B. B receives the input $(\texttt{sign}, C_B, tid)$ and message m_1 and verifies whether the received contract text C_A is identical to the input contract text C_B. If not or if wakeup is input before m_1 is received, the players disagree about the contract and B returns $(\texttt{rejected}, tid)$. Else, it signs the received message and sends it as $m_2 := \mathrm{sig}_{\mathsf{B}}(m_1)$ to A. If it received m_2, A then signs the received message again, sends it as $m_3 := \mathrm{sig}_{\mathsf{A}}(m_2)$ back and outputs $(\texttt{signed}, C, tid)$. On receipt of message m_3, B outputs $(\texttt{signed}, C, tid)$ as well. After a successful execution of this optimistic protocol, A and B store m_3 under the tid for later use in a verification protocol.

If A gets an input wakeup before receiving message m_2, it starts "$\texttt{resolve}_1$" to abort the protocol. If B did not receive message m_3, it may be that A nevertheless was able to compute a valid contract m_3 after receiving m_2. Therefore B starts the "$\texttt{resolve}_2$"-protocol to invoke the third party to guarantee fairness.

Recovery from Exceptions (Sub-Protocol "$\texttt{resolve}_1$"): To start this recovery protocol, A sends the message $m_{4A} := \mathrm{sig}_{\mathsf{A}}(\texttt{abort})$ to T. If the third party made a decision before, it resends the decision. If the third party is in its start-state, it changes to the aborted state and acknowledges this to A by sending $m_5 := \mathrm{sig}_{\mathsf{T}}(\texttt{aborted})$. If A receives m_5, it outputs $(\texttt{rejected}, tid)$. If A receives m_5', it outputs $(\texttt{signed}, C, tid)$.

Recovery from Exceptions (Sub-Protocol "$\texttt{resolve}_2$"): First, B sends $m_{4B} := \mathrm{sig}_{\mathsf{B}}(m_2)$ containing m_1 and m_2 to the third party T. If the protocol was aborted, the third party resends m_5. Else, it sends an affidavit $m_5' := \mathrm{sig}_{\mathsf{T}}(m_2)$ to B and changes to the signed–state. If B receives m_5', it outputs $(\texttt{signed}, C, tid)$. If it receives m_5, it outputs $(\texttt{rejected}, tid)$.

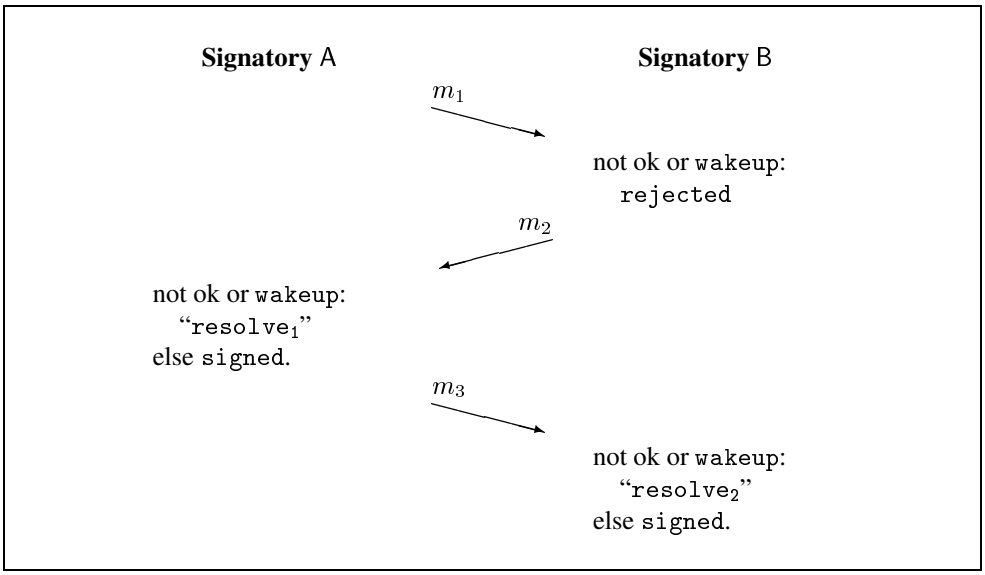

Figure 18: Optimistic Behavior of the Message-Optimal Synchronous Scheme 6.

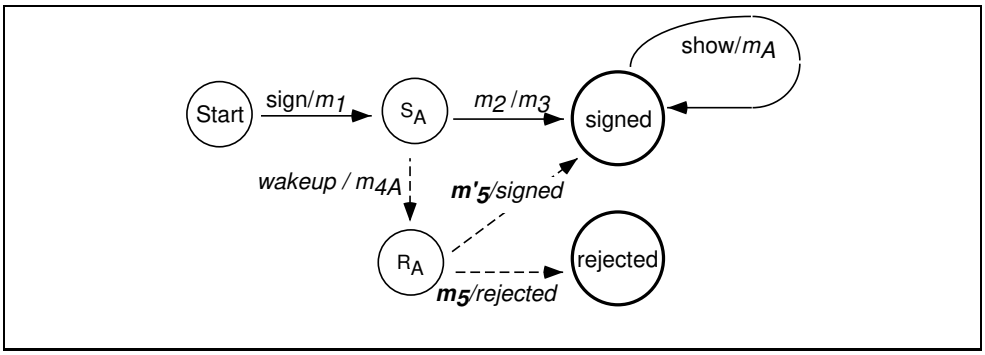

Figure 19: Signatory A of Scheme 6.

*Verification of a Contract (Protocol "*show*"):* On input (show, tid), signatory A looks up m_2 and sends it as $m_A := \mathrm{sig}_A(m_2)$ to the verifier. The verifier then forwards this message to the third party. If the third party resends an abort message m_5, the verifier outputs rejected. If the third party answers with m'_5, it outputs signed.

Signatory B on the other hand either looks up m'_5 or m_3 signs it and and sends it to the verifier. If the verifier receives a correct message, it outputs signed and rejected, else.

□

Note that m_3 is a valid contract for B in any case whereas it need not be a valid contract for A if it aborted the protocol.

Lemma 7 (Security of Scheme 6)
Scheme 6 is a asynchronous fair contract signing scheme with three-party verification which is optimistic on agreement. ◇

Proof of Lemma 7: The scheme adheres to Definition 4 by construction. We now show that each of the requirements described in Definitions 3 and 6 are fulfilled:

Correct Execution: If both correct players A and B input (sign, C, tid) with identical tid and C, then both receive a valid contract m_3 and output (signed, C, tid). If the contracts or tid's differ and a signatory, say A, inputs wakeup, this player will start resolve by sending m_{4A} and will output (rejected, tid) after receiving m_5.

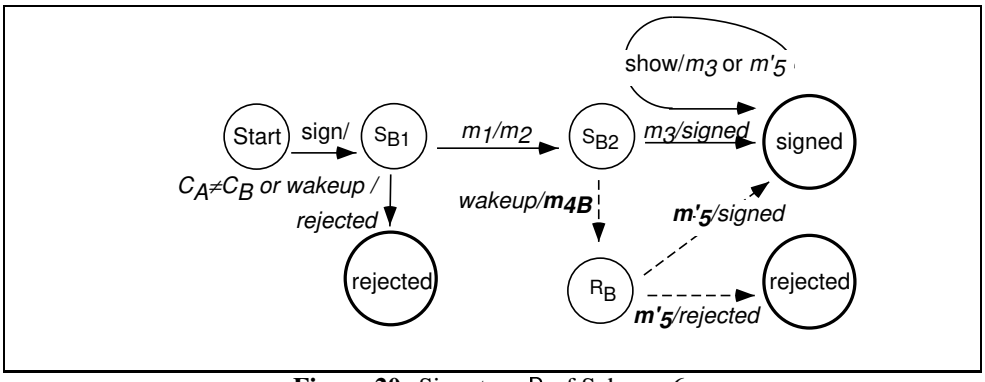

Figure 20: Signatory B of Scheme 6.

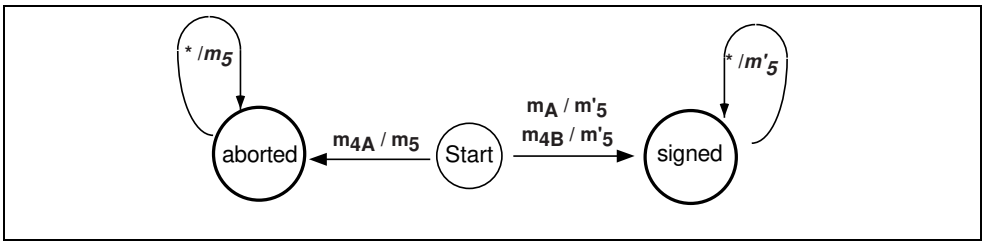

Figure 21: Third Party T of Scheme 6.

Unforgeability of Contracts: In order to convince a correct verifier V for a given tid, one needs correct messages m_3 or m'_5 for this tid. Since m_3 as well as m'_5 contain signatures from both participants, a correct signatory input $(\texttt{sign}, C, tid)$.

Verifiability of Valid Contracts: If a correct signatory A outputs $(\texttt{signed}, C, tid)$ then it received m_2 or m'_5 from which it can extract m_2 which is a valid contract if the third party does not resend m_5. If m'_5 has been received and the third party resends m_5, the third party is incorrect. If m_2 has been received and a correct third party resends m_5, A is incorrect since it sent m_{4A} while receiving m_2. If B output $(\texttt{signed}, C, tid)$, it received m_3 or m'_5 which will be accepted by the verifier in any case.

No Surprises with Invalid Contracts: Let us first assume that a correct signatory A returned `rejected` on input $(\texttt{sign}, C, tid)$ whereas B is able to convince the verifier. This requires that B knows m_3 or m'_5 for the given tid and C. Since A returned `rejected`, it executed "$\texttt{resolve}_1$" receiving m_5. Thus, B did not receive m'_5 from the correct T. If B received m_3, A was incorrect since it sent m_3 while executing "$\texttt{resolve}_1$".

If B returned `rejected` whereas A is able to convince a verifier, A knows m_A which contains m_2. Therefore, B output `rejected` after receiving m_5 during "$\texttt{resolve}_2$" and this decision will be resent during recovery and a correct verifier will decide on `rejected`.

Termination on Asynchronous Network: The scheme requires at most time 2 after an input `wakeup`.

Optimistic on Agreement on Asynchronous Network: If two correct signatories input $(\texttt{sign}, C, tid)$ and both agree, signatory A outputs $(\texttt{signed}, C, tid)$ after time 2 and player B after time 3.

■

Theorem 7 (Optimality of Scheme 6)
There exists no asynchronous optimistic contract signing scheme with three-party verification with a "`sign`"–protocol with less than three messages in case of agreement and every

three-message protocol requires at least time 3. ◇

Proof of Theorem 7: If we assume that there would be a two-message asynchronous optimistic "`sign`"–protocol, unforgeability requires that each signatory sends one of these messages. Furthermore, each of these messages must be a valid contract if the signatures for the given tid have not been revoked. Optimism requires that if this single message is received correctly and the signatories agree, this signatory outputs `signed`.

Let us assume that one signatory receives `wakeup` after sending its message but before receiving the message from the peer. If this is the first request to T for this tid, this signatory is required to recover with the third party to `rejected` in order to guarantee unforgeability. But this contradicts the no-surprises requirement since the other signatory already output `signed`.

Let us assume that there would be an optimistic "`sign`"–protocol with three messages in two rounds. Then this can be used to construct a two-message protocol by shoving messages like in the proof of Theorem 3 but such a two-message protocol does not exist. ∎

9 Conclusion

We described new and existing protocols for fair contract signing on synchronous and asynchronous networks. We have proven tight bounds for different network and contract signing models.

One conclusion is that in practice, optimistic protocols should be better for most applications since a high percentage of faulty protocol executions seems unlikely. Optimistic protocols have practical advantages such as a higher availability and more privacy against the third party, which requires additional effort (see, e.g., [FrRe 97]) for in-line protocols.

10 Acknowledgment

The authors would like to thank Hans-Jürgen Guth for helpful discussions about the problem of the half-sold house and N. Asokan for valuable comments about timing of asynchronous protocols. This work was partially supported by the ACTS SEMPER project <www.semper.org>; however, it represents the view of the authors only.

References

[AsSW2 96] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Multi-Party Fair Exchange; IBM Research Report RZ 2892, IBM Zurich Research Laboratory, Zürich, November 1996.

[AsSW 97] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Fair Exchange; 4th ACM Conference on Computer and Communications Security, Zürich, April 1997, 6-17.

[AsSW3 97] N. Asokan, Victor Shoup, Michael Waidner: Asynchronous Protocols for Optimistic Fair Exchange; IBM Research Report RZ 2976, IBM Zurich Research Laboratory, Zürich, November 1997.

[BGMR 90] M. Ben-Or, O. Goldreich, S. Micali, R. L. Rivest: A Fair Protocol for Signing Contracts; IEEE Transactions on Information Theory 36/1 (1990) 40-46.

[Blum 81] Manuel Blum: Three Applications of the Oblivious Transfer; Version 2: September 18, 1981, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkley, Ca. 94720.

[Blu2 83] Manuel Blum: How to Exchange (Secret) Keys; ACM Transactions on Computer Systems 1/2 (1983) 175-193.

[BüPf 89] Holger Bürk, Andreas Pfitzmann: Digital Payment Systems Enabling Security and Unobservability; Computers & Security 8/5 (1989) 399-416.

[BüPf 90] Holger Bürk, Andreas Pfitzmann: Value Exchange Systems Enabling Security and Unobservability; Computers & Security 9/8 (1990) 715-721.

[Damg 95] Ivan Bjerre Damgård: Practical and Provably Secure Release of a Secret and Exchange of Signatures; Journal of Cryptology 8/4 (1995) 201-222.

[Even 83] Shimon Even: A Protocol for Signing Contracts; ACM SIGACT News 15/1 (1983) 34-39.

[EvGL 85] Shimon Even, Oded Goldreich, Abraham Lempel: A Randomized Protocol for Signing Contracts; Communications of the ACM 28/6 (1985) 637-647.

[EvYa 80] Shimon Even, Yacov Yacobi: Relations Among Public Key Signature Systems; Technical Report Nr. 175, March 1980, Computer Science Department, Technion, Haifa, Israel.

[FrRe 97] Matthew K. Franklin, Michael K. Reiter: Fair Exchange with a Semi-Trusted Third Party; 4th ACM Conference on Computer and Communications Security, Zürich, April 1997, ACM Press, New York 1997, 1-5.

[Gold 83] Oded Goldreich: A simple protocol for signing contracts; Crypto '83, Plenum Press, New York 1984, 133-136.

[GoMR 88] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM Journal on Computing 17/2 (1988) 281-308.

[Gong2 95] Li Gong: Efficient Network Authentication Protocols: Lower Bounds and Optimal Implementations; Distributed Computing 9/3 (1995) 131-145.

[Lamp 78] Leslie Lamport: Time, Clocks, and the Ordering of Events in a Distributed System; Communications of the ACM 21/7 (1978) 558-565.

[Lync 96] Nancy A. Lynch: Distributed Algorithms; Morgan Kaufmann, San Francisco 1996.

[Mica 97] Silvio Micali: Certified E-Mail with Invisible Post Offices - or - A Low-Cost, Low-Congestion, and Low-Liability Certified E-Mail System; presented at RSA 97; sent by S. Micali to Asokan and M. Schunter, May 2, 1997.

[PeSL 80] Marshall Pease, Robert Shostak, Leslie Lamport: Reaching Agreement in the Presence of Faults; Journal of the ACM 27/2 (1980) 228-234.

[PfSW 98] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Optimal Efficiency of Optimistic Contract Signing; to appear at 17th Symposium on Principles of Distributed Computing (PODC), ACM, New York 1998.

[RSA 78] R. L. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of the ACM 21/2 (1978) 120-126, reprinted: 26/1 (1983) 96-99.

[SiKS 97] A. Silberschatz, H. Korth, S. Sudarshan: Database system concepts. McGraw-Hill, 1997.

[Tel 91] Tel, Gerard: Topics in Distributed Algorithms. Cambridge University Press, Cambridge 1991.

[ZhGo 97] Jianying Zhou, Dieter Gollmann: An Efficient Non-repudiation Protocol; 10th Computer Security Foundations Workshop, IEEE Computer Society Press, Los Alamitos 1997, 126-132.