Optimal Efficiency of Optimistic Contract Signing

Birgit Pfitzmann
Universität des Saarlandes
<pfitzmann@cs.uni-sb.de>

Matthias Schunter
Universität Dortmund
<schunter@acm.org>

Michael Waidner
IBM Zurich Research Laboratory
<wmi@zurich.ibm.com>

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.

1 Introduction

A *contract* is a non-repudiable agreement on a given text [4]. 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 [4], 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.

To appear in 17th Symposium on Principles of Distributed Computing (PODC), ACM, New York 1998.
Preprints can be obtained at <`www.semper.org/sirene/`>.

In order to minimize this involvement while guaranteeing fairness, the concept of so called "optimistic" protocols has been introduced [1, 3][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 [4]. In [11], it was shown that no deterministic contract signing scheme (called "public-key agreement system" in [11]) without third party exists if the verifier is stateless 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* [4, 5, 10, 12] (see [8] 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 [3], 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 [9]. The first optimistic scheme in our sense is based on *gradual increase of privilege* [3]: 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 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.

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

[2] See also [6, 7] for optimistic protocols for payment for receipt or goods, or [16, 22] 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.

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 [3]. 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: [1] describes a synchronous contract signing protocol with four messages. This was improved in [2] 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 for atomicity of distributed transactions [20], 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 [17]: 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. Table 1 gives a detailed summary of our results: We present a message- (3 messages) and a round-optimal (2 rounds) synchronous optimistic contract signing schemes as well as a time-optimal (time 3) asynchronous scheme based on an arbitrary digital signature scheme. We prove the optimality of these new schemes as well as the optimality of the scheme described in [2] 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.

Model			Our Results			
C	T	Op	t	m	Optimal	Proof
s	sl	+	3	**3**	Scheme 1	Theorem 1
s	sl	+	**2**	4	Scheme 2	Theorem 1
a	sl	+	Impossible			Theorem 4
a	sk	(+)	4	**4**	See [2]	Theorem 2
a	sk	(+)	**3**	6	Scheme 3	Theorem 3
a	sk	–	**2**	**4**	Scheme 4	Theorem 5

Legend:

- C Communication Model: "s" for synchronous, "a" for asynchronous.
- T Properties of the third party: "sl" for state-less, "sk" for state-keeping.
- Op "+" stands for optimistic protocols (Def. 3), "(+)" stands for optimistic on agreement (Def. 4), and "–" stands for non-optimistic protocols (Def. 2).
- t Time for the output of a contract if the signatories agree (Bold figures are provably optimal).
- m Number of messages in case of agreement.

Table 1: Provably Optimal Schemes By Model.

Finally, we prove the optimality of a well-known asynchronous non-optimistic scheme (4 messages in time 2) in order to enable us to substantiate our claim that optimistic protocols are more efficient than non-optimistic schemes if exceptions and misbehavior are unlikely.

2 Definitions

2.1 Network Models and Protocol Complexity

We distinguish between the "standard" synchronous and asynchronous network models [15, 21]. 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.

2.2 Contract Signing

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

Definition 1 (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 stateless, i.e., has no memory between subsequent protocol runs. The algorithms A and B are called signatories,

[4] For similar research on authentication protocols see [14].

[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.

and V is called verifier. The algorithms can carry out two interactive protocols:

Contract Signing (Protocol "sign"): Each signatory $X \in \{A, B\}$ obtains a local input ($\mathtt{sign}$, 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: ($\mathtt{signed}$, C, tid) containing a contract text C or ($\mathtt{rejected}$, 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 ($\mathtt{show}$, tid). A does not make a local output. The verifier V outputs either ($\mathtt{signed}$, C, tid) or ($\mathtt{rejected}$, tid).

□

Intuitively, an output ($\mathtt{signed}, C, tid$) 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 ($\mathtt{rejected}$, 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 ($\mathtt{wakeup}, tid$), 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 2 (Fair Contract Signing)
A contract signing scheme (Def. 1) 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 ($\mathtt{sign}$, C_A, tid) to A and ($\mathtt{sign}$, 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 ($\mathtt{signed}$, C_A, tid) iff $C_A = C_B$ or else ($\mathtt{rejected}, tid$) to both signatories if none inputs wakeup.

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

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

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

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

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

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 subprotocol called "resolve".

Definition 3 (Optimistic Contract Signing)
A fair contract signing scheme (Def. 2) 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 ($\mathtt{wakeup}$, 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 4 (Optimistic on Agreement)
A fair contract signing scheme (Def. 2) 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 ($\mathtt{sign}$, 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 ($\mathtt{sign}$, C, tid) with a fresh and unique tid and a $C \in M$ and do not input ($\mathtt{wakeup}$, tid), the third party does not send or receive messages in the "sign"-protocol.

[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 the full report [18].

□

2.4 Notations and Assumptions

All our schemes are based on a secure digital signature scheme [13, 19] 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, Ⓐ $\stackrel{a/b}{\to}$ Ⓑ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 $\neg a$.

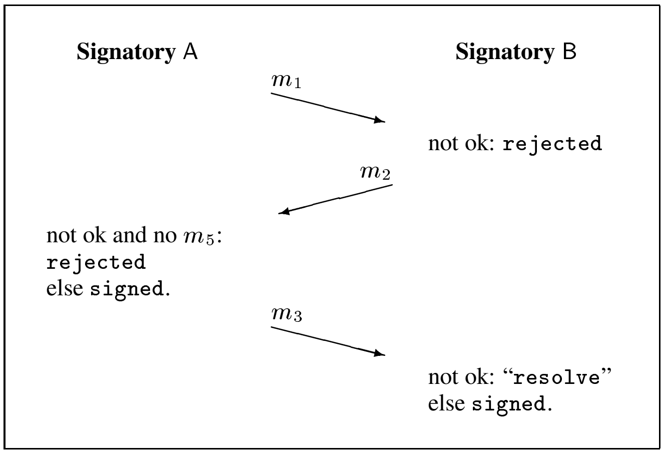

Figure 1: Optimistic Behavior of the Message-Optimal Synchronous Scheme 1.

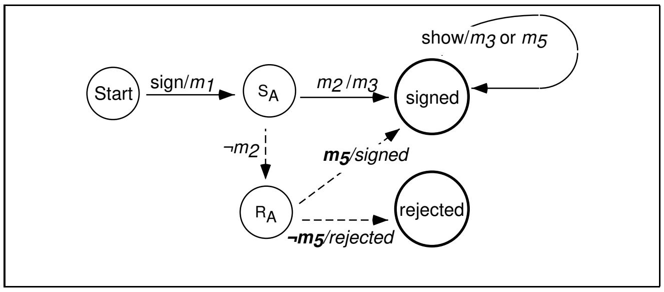

Figure 2: Signatory A of Scheme 1.

3 A Message-Optimal Synchronous Scheme

Our message-optimal optimistic scheme[8] for synchronous networks requires three messages in the optimistic case using a stateless 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 (A, B, 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 ($\mathtt{sign}$, 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 ($\mathtt{sign}, C_B, tid$) 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 ($\mathtt{rejected}, tid$). 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 ($\mathtt{signed}$, C, tid). On receipt of message m_3, B outputs ($\mathtt{signed}, C, tid$) 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.

[8] The message flows are similar to the optimistic protocol in [16] 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 [1].

If A does not receive message m_2 it waits until Round 5, and, if m_5 was not received, it outputs ($\mathtt{rejected}, tid$). 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 ($\mathtt{signed}, C_A, tid$). Furthermore T sends an affidavit on m_2 in $m_5' := \mathrm{sig}_{\mathsf{T}}(m_2)$ to B and B outputs ($\mathtt{signed}, C, tid$). 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 ($\mathtt{show}$, tid), a signatory looks up m_3 or m_5' and sends it to the verifier. The verifier verifies it and outputs ($\mathtt{signed}$, C, tid) if this succeeds and ($\mathtt{rejected}$, tid) else.

□

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

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

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

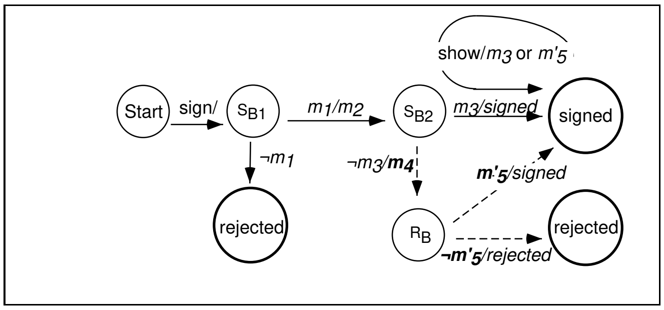

Figure 3: Signatory B of Scheme 1.

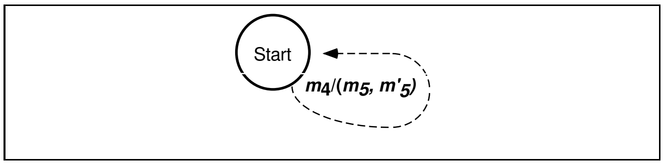

Figure 4: Third Party T of Scheme 1.

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 $(\mathtt{sign}, C, tid)$.

Verifiability of Valid Contracts: If A outputs $(\mathtt{signed},\ C,\ tid)$ 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 $(\mathtt{signed},\ C,\ tid)$ 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 $(\mathtt{sign}, C, tid)$ 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 $(\mathtt{sign},\ C,\ tid)$, A outputs $(\mathtt{signed},\ C,\ tid)$ after round 2 whereas player B outputs $(\mathtt{signed}, C,\ tid)$ after round 3. If they disagree, i.e., input different contracts, A outputs $(\mathtt{rejected},\ tid)$ 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, there is no optimistic contract signing scheme with less than 3 messages in the optimistic case and a 3-message protocol needs at least 3 rounds. ◇

Proof of Theorem 1: Let us assume that there exists an optimistic contract signing scheme where the optimistic case needs three messages in two rounds. 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. ■

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 $(\mathtt{sign}, C_A,\ tid)$ 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 $(\mathtt{rejected}, tid)$ 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 $(\mathtt{signed}\ C_A,\ tid)$. 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 $(\mathtt{signed},\ C,\ tid)$. Else, they output $(\mathtt{rejected}, tid)$.

Verification of a Contract (Protocol "`show`"): On input $(\mathtt{show}, tid)$, 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 $(\mathtt{rejected}, tid)$ and else $(\mathtt{signed}, C,\ tid)$.

□

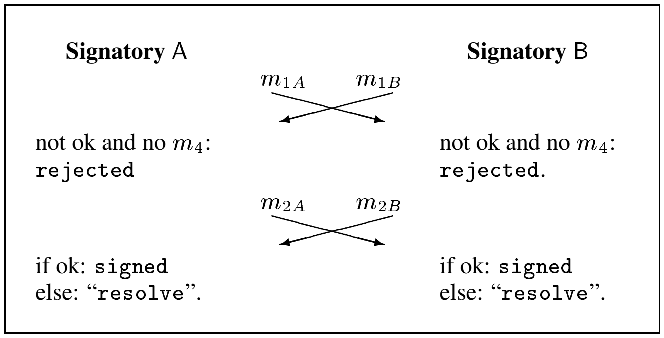

Figure 5: Optimistic Behavior of the Round-Optimal Synchronous Scheme 2.

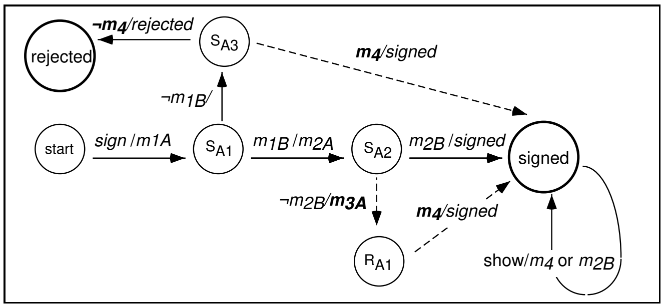

Figure 6: Signatory, e.g., A, of Scheme 2.

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

Proof of Lemma 2: The scheme adheres to Definition 1 by construction. We now show that it fulfills the requirements stated in Definitions 2 and 3:

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 $(\texttt{signed}, C, tid)$ 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 $(\texttt{rejected}, tid)$.

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`, she 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.

■

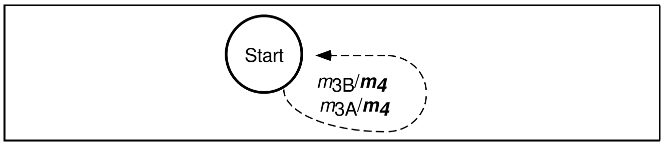

Figure 7: Third Party T of Scheme 2.

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 [2]. 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).

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.

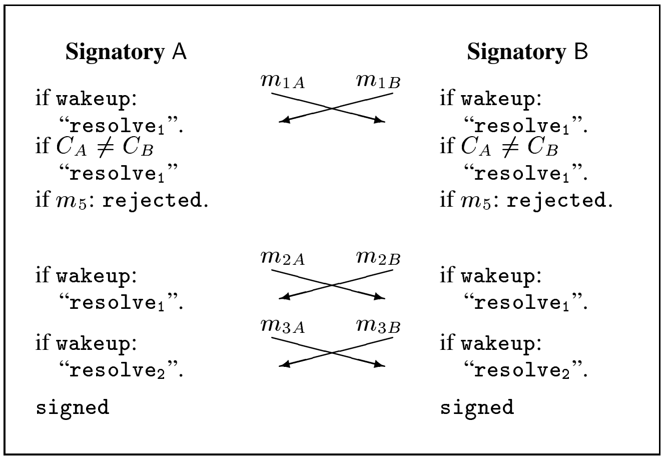

Figure 8: Optimistic Behavior of the Time-Optimal Asynchronous Scheme 3.

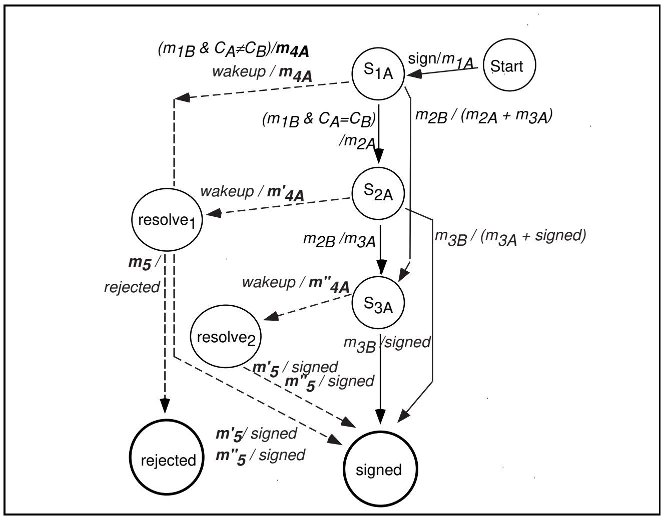

Figure 9: Signatory, e.g., A, of Scheme 3 (states S_{2A} or S_{3A} may be bypassed if messages are reordered).

Verification of a Contract (Protocol "show"*):* After the input (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 (rejected, tid) and else (signed, C, tid).

□

Lemma 3 (Security of Scheme 3)
Scheme 3 is a fair optimistic contract signing scheme. ◇

Proof of Lemma 3: Scheme 3 adheres to Definition 1 by construction. We now show that it also fulfills the requirements stated in Definitions 2 and 4.

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

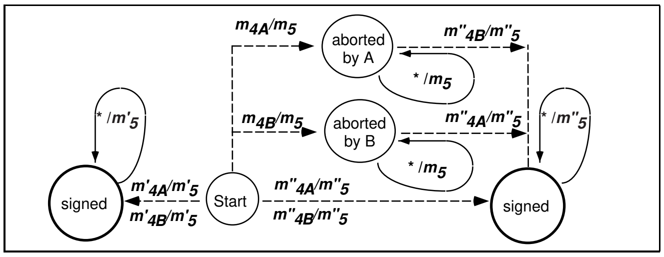

Figure 10: Third Party T of Scheme 3.

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 [2])
There is no asynchronous optimistic contract signing scheme with three messages in the optimistic case. ◇

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. ◇

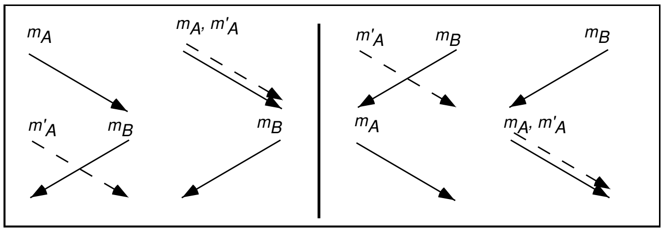

Figure 11: Proof of Theorem 3: Saving a Dashed Message by Shoving it Up or Down.

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 asks the other signatory, the third party could not 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 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 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. If it now decides on `rejected` based on m_1 only, an incorrect A could make a contract invalid after a successful completion that did not involve the third party (i.e., after sending m_3). If the third party decides on `signed` since B may later recover, then a valid contract could be obtained without B's participation. ∎

This enables us to prove the optimality of Scheme 3:

Theorem 3 (Optimality of Scheme 3)
There is no asynchronous optimistic contract signing scheme in time 2 in the optimistic case and every 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 optimistic contract signing scheme with state-less 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 $TP(m_1, \ldots, m_{k-1}) := TP(m_1, \ldots, m_k) = \texttt{signed}$. 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.

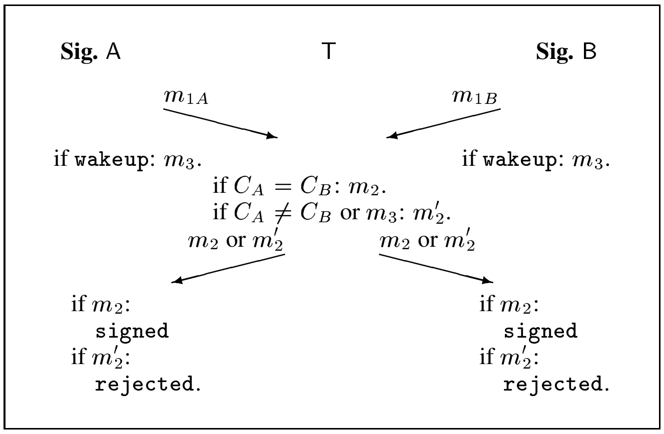

Figure 12: Behavior of the Optimal Asynchronous Scheme 4 with In-Line Third Party.

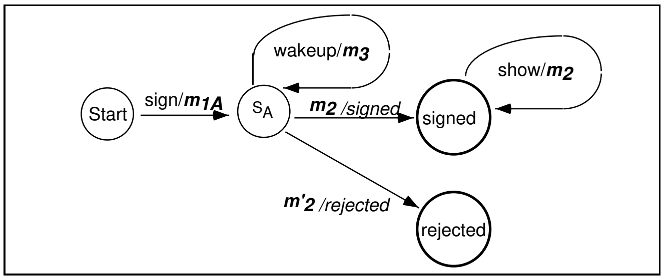

Figure 13: Signatories, e.g., A, of Scheme 4.

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 $(\texttt{sign}, C, tid)$, each signatory, say A, sends a signed message $m_{1A} := \mathrm{sig}_{\mathsf{A}}(C_A)$ containing the contract text C_A 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 a fair contract signing scheme if the machine T is correct. ◇

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

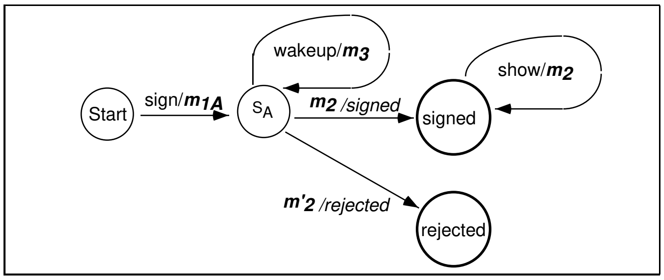

Figure 14: Third Party T of Scheme 4.

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 $(\texttt{wakeup}, tid)$, the scheme requires at most time 2 to output `signed` or `rejected`.
■

We now prove the optimality of Scheme 4.

Theorem 5 (Optimality of Scheme 4)
There exists no asynchronous contract signing protocol with three messages or only one round in the fault-less case. ◇

Note that this shows that one cannot improve the efficiency of the optimistic protocol presented in [2] by allowing the third party to participate in the "`sign`"–protocol.

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 recieve any messages, the protocol is optimistic and a three message optimistic protocol 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 since 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.

If we assume that the messages are sent in the sequence A, B, T, A then B must be able to win a dispute with its only message received. If B then does not send its message to T. If we assume that `wakeup` was input to A, A must be able to recover to `signed` with its only message sent. However, in this case, A could as well recover with the first message without sending it thus passing verification without B. This would contradict the unforgeability requirement for B. Recovery to `rejected` is also not possible since it contradicts the "no surprises" requirement for A.

If we assume that the sequence is A, T, B, A then B received a valid contract with the message from T. Since B may omit the last message, A would be required to pass a verification with its initial message (maybe after recovering together with T). This contradicts the unforgeability requirement since A and T would be able to end up with a valid contract without input from B (Remember: We ruled out three-party disputes).

To prove that no protocol in time 1 exists, we assume there would be 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 disputes. Therefore, these messages can be omitted. The resulting protocol would be a two-message protocol where each message is a valid contract. If one player, say B, does not send its message, A must be allowed to recover to `signed` without any signed messages from B. This contradicts the unforgeability requirement. ■

7 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 comments about timing of asynchronous protocols. Furthermore, we would like to thank the anonymous reviewers for valuable comments.

References

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

[2] N. Asokan, V. Shoup, M. Waidner: Asynchronous Protocols for Optimistic Fair Exchange; IBM Research Report RZ 2976, IBM Zurich Research Laboratory, Zürich, November 1997.

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

[4] M. Blum: Three Applications of the Oblivious Transfer; Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, September 18, 1981.

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

[6] H. Bürk, A. Pfitzmann: Digital Payment Systems Enabling Security and Unobservability; Computers & Security 8/5 (1989) 399-416.

[7] H. Bürk, A. Pfitzmann: Value Exchange Systems Enabling Security and Unobservability; Computers & Security 9/8 (1990) 715-721.

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

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

[10] S. Even, O. Goldreich, A. Lempel: A Randomized Protocol for Signing Contracts; Communications of the ACM 28/6 (1985) 637-647.

[11] S. Even, Y. Yacobi: Relations Among Public Key Signature Systems; Technical Report Nr. 175, Computer Science Department, Technion, Haifa, Israel, March 1980.

[12] O. Goldreich: A simple protocol for signing contracts; Crypto '83, Plenum Press 1984, 133-136.

[13] S. Goldwasser, S. Micali, R. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM Journal on Computing 17/2 (1988) 281-308.

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

[15] N. Lynch: Distributed Algorithms. Morgan Kaufman 1996.

[16] S. Micali: Certified E-Mail with Invisible Post Offices; 6th Annual RSA Data Security Conference; January 1997.

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

[18] B. Pfitzmann, M. Schunter, M. Waidner: Optimal Efficiency of Optimistic Contract Signing; IBM Research Report RZ 2994 (#93040), IBM Zurich Research Laboratory, Zürich, 03/02/1998.

[19] R. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of the ACM 21/2 (1978) 120-126.

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

[21] G. Tel: Topics in Distributed Algorithms. Cambridge University Press 1991.

[22] J. Zhou, D. Gollmann: An Efficient Non-repudiation Protocol; 10th Computer Security Foundations Workshop, IEEE Computer Society Press 1997, 126-132.