Multiparty Contract Signing over a Reliable Network

Simona Orzan [1] Erik de Vink [2]

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O.Box 513, 5600 MB Eindhoven, The Netherlands
s.m.orzan@tue.nl, evink@win.tue.nl

Abstract

Most contract signing protocols make use of a trusted third party (TTP) to ensure fairness. It has been shown that in the crash network model, this is inevitable. However, for stronger networks, where misbehavior is allowed but failure excluded, the necessity of a TTP has not yet been debated. We consider a strong network model, that includes reliable broadcast, bounded delays and timestamps and use it to describe a simple multiparty contract signing protocol that does not rely on a TTP. This shows that by strengthening the assumptions on the network, the transfer of trust from one dedicated server to the network is feasible. The result is commented in a more general setting of multiparty protocols and problems. The correctness of the proposed protocol for any number of participants is proven using process algebra techniques.

1 Introduction

Signing a contract in real life simply requires some people in a room, a pen and a piece of paper. Signing a digital contract amongst multiple partners over a network, though, requires a lot more attention and well-chosen security measures to protect against possible misbehavior or network failures. A multiparty contract signing protocol is a series of messages that, if followed precisely, implements this complex digital operation and ensures desirable properties. The most important property is *fairness*. In a fair protocol, either each well-behaving participant obtains the signed contract, or nobody does.

In this paper we address two issues regarding contract signing protocols. The first is the presence of a Trusted Third Party (TTP). Contract signing

[1] Also affiliated to CWI, Amsterdam.
[2] Also affiliated to LIACS, Leiden University.

protocols usually rely on such a TTP, which can be contacted offline or online. However, a TTP is clearly a bottleneck, both in performance and trust, and constitutes a single point of failure. Solutions without a TTP include protocols with gradual information exchange, randomized and probabilistic protocols, all with a non-zero failure probability. It has also been proved that in the so-called crash network model, where connections are faulty and parties possibly misbehaving, the problem is impossible to solve without a TTP [15]. In other, more reliable network models, the contract signing problem has not been studied, basically because reliable networks are not considered realistic. But in the context of fault-tolerant software architectures and coordination middleware platforms [14,18,17], a reliable communication medium is plausible. Therefore the question arises whether these generic models can take over the functionality of the dedicated TTP, and if so, which communication primitives are necessary/sufficient.

The second issue concerns the formal analysis of multiparty contract signing protocols and fair exchange protocols in general. We want to formally prove security properties of multiparty contract signing protocols parametric in the number of participants. Referring to the well-known quote of Roger Needham about three-line security protocols [1], it comes without saying that formal verification is important in this field. Even more so for multiparty protocols that come with the burden of non-intuitive scheduling and state explosion. Although these two questions don't seem related at first, they are subtly linked. Namely, the protocols without a TTP exhibit more symmetry, and are therefore more likely in reach of recent parametrized verification techniques.

We seek to answer our two-fold question by strengthening the network to the point where the network itself can provide the functionality of a trusted party. So, conceptually, trust is transferred from the TTP to the communication structure. Identifying the precise point where this transfer is complete remains an open challenge, but we give a partial solution here. The proposed network model includes (i) reliable broadcast with bounded delay, (ii) timestamps, and (iii) a proof-of-send mechanism. The latter mechanism provides the sender of a message with an unforgeable proof of delivery of the message to the network. Admittedly, the resulting protocol is of limited practical value; the assumptions on the network are strong. However, it shows that a TTP-free solution for the multiparty contract signing problem exists and is, moreover, not dependent on the number of misbehaving parties. The proposed protocol is also simple enough to be in reach of algebraic verification techniques. We manage to prove its fairness for any number of participants. The proof is of an inductive flavor, having equivalences checked automatically for the base case, and using axioms of the process algebra μCRL to lift the equivalences to the general case.

As far as we know, no protocol similar to the one presented here exists in the literature. The closest related work is reported in [7]. In the context of parametric systems, μCRL has been used before [12,16]. The proofs presented there are however not directly comparable to ours. Other verification techniques have been applied to small scenarios of MPCS protocols [6].

Overview We first introduce the problem of multiparty contract signing (Section 2), then our simple protocol (Section 3) together with a formal correctness proof (Section 4). We then discuss in detail the implications and challenges of transferring trust from a server to the network (Section 5), and finish with some conclusions in Section 6.

2 Multiparty Contract Signing

The informal idea of what constituted a multiparty contract signing protocol (MPCS protocol) and what it should achieve has broad consensus. In a nutshell, there are N participants, some possibly malicious, who should reach agreement on whether a contract that they've been negotiating will be signed or aborted. Moreover, if it is signed, all the participants should be able to show the signed contract to an outside observer, also called the verifier. Below, we describe this more formally and list the expected properties. This also indicates how such a protocol can go wrong.

Let $\mathtt{P}_1 \cdots \mathtt{P}_N$ be N parties willing to sign a contract $\mathtt{c}$ and $\mathtt{Verifier}$ the outside observer. A MPCS protocol consists of two subprotocols:

$$\begin{array}{lcl} \mathtt{SIGN}\ (\mathtt{c}, \mathtt{P}_1 \cdots \mathtt{P}_N) : \langle \mathrm{d}_1 \cdots \mathrm{d}_N \rangle & \longmapsto & \mathrm{out}_1 \cdots \mathrm{out}_N \\ \mathtt{VERIFY}\ (\mathtt{P}_i, \mathtt{Verifier}) : \mathrm{out}_i & \longmapsto & \{\mathsf{accept}, \mathsf{reject}\}. \end{array}$$

The first starts from the individual decisions d_i (sign or abort) and works towards reaching agreement. The outcome is a value out_i for every player $\mathtt{P}_i$. This is either the signed contract, or the abort decision. The $\mathtt{VERIFY}$ subprotocol describes the interaction between a player and an external verifier. The verifier will examine the signed contract that $\mathtt{P}_i$ presents and will output the decision accept only if it is a valid signed contract. Ideally, an MPCS protocol guarantees the following:

- *Fairness*: At the end of the protocol, either every signer obtains all the other signers' signatures on the contract, or nobody gets all the other signers' signatures.
- *Timeliness*: There exists some mechanism to ensure termination, both for the signing and the verification subprotocol.
- *Abuse-freeness*: No signer should be able to prove to an external observer that he has the power to choose between concluding the protocol successfully and aborting it.

As a malicious participant we consider a participant who does not follow the protocol, but tries to make it fail instead.

3 A simple MPCS protocol

We now describe a straightforward multiparty contract signing protocol without TTP in which N participants aim to sign a contract $\mathtt{c}$ together. The participants are geographically distant and communicate by broadcasting messages over an asynchronous network. They know each other and it is assumed that the discussions and negotiations on $\mathtt{c}$ have taken place previously, outside the contract signing protocol. Every participant i is capable of constructing an *unforgeable signature* $\mathrm{sign}(i, x)$ on any binary content x. This is usually implemented via a public-key infrastructure, which is assumed to be in force here. We follow the usual blackbox interpretation of cryptography and abstract away from its details.

The network model

In order to eliminate the need for a trusted party, we strengthen the communication model. In particular, there are three key features that we require extra from the network:

- It supports *reliable broadcast* as communication primitive, meaning that messages don't get lost (although the order doesn't have to be preserved) and messages are delivered within a bounded delay.
- It automatically *timestamps* all messages with the time of the send action. A global clock is not necessary, but we do assume that there is some loose synchronization between the local clocks of the participants' hosts such that the maximum difference between two clocks is always bounded by a constant known to all participants.
- It has the capability to provide unforgeable *proofs of send* to anybody who places a message on the broadcast channel. Whenever participant i broadcasts a message m, the network gives back to i a timestamped proof-of-send which is simply an unforgeable copy of the actual network message $\mathbf{msg}\langle i, m, \mathtt{t}\rangle$.

Note that the network has no special knowledge regarding MPCS protocols. It is just generic support, for example provided by the middleware, rather than a 'consciously' participating party. For more comments on this model, see Section 5.

The protocol

We will denote broadcast by $\Rightarrow$ and receiving a message by $\leftarrow$. The protocol for participants $\{\mathtt{P}_1 \cdots \mathtt{P}_N\}$ is described in Fig. 1. Before running the SIGN protocol, the participants establish agreement on the text of the contract $\mathtt{c}$ and on a value Δ which indicates the maximum time that a participant is allowed to ponder about whether to sign or not. The value Δ takes into account a possible skew of the local clocks.

$\texttt{SIGN}\,(\{\texttt{P}_1 \cdots \texttt{P}_N\},\, \texttt{c},\, \Delta) \;\longmapsto\; \text{out}_1 \cdots \text{out}_N$

- for all $i \in \{1 \cdots N\}$, $\texttt{P}_i \Rightarrow :\ \text{sign}(i, \texttt{c})$
- for all $i, j \in \{1 \cdots N\}\ (i \neq j)$, $\texttt{P}_i \leftarrow :\ \mathbf{msg}\langle j, \text{sign}(j, \texttt{c}), \texttt{t}'\rangle$
- $\texttt{P}_i$ leaves the protocol when
  · she has successfully sent and received all the messages, or
  · a timeout occurs, or
  · she decides to abort
- the outcome of the algorithm for $\texttt{P}_i$ is

$$\text{out}_i \stackrel{\text{def}}{=} (\text{out}_{i,1} \cdots \text{out}_{i,N}),$$

$$\text{where} \begin{cases} \text{out}_{i,i} \stackrel{\text{def}}{=} \mathbf{msg}\langle i, \text{sign}(i, \texttt{c}), \texttt{t}\rangle \text{ or } \mathsf{abort} \\ \text{out}_{i,j} \stackrel{\text{def}}{=} \mathbf{msg}\langle j, \text{sign}(j, \texttt{c}), \texttt{t}_j\rangle \text{ or } \mathsf{timeout} \text{ for } i \neq j \end{cases}$$

$\texttt{VERIFY}\,(\{\texttt{P}_1 \cdots \texttt{P}_N\},\, \texttt{c},\, \Delta,\, \text{out}_i) \;\longmapsto\; \mathsf{accept} \text{ or } \mathsf{reject}$

if $\text{out}_{i,i} \neq \mathbf{msg}\langle \text{sign}(i, \texttt{c}), \texttt{t}\rangle$ then reject

else if $\exists j \neq i :\ \text{out}_{i,j} = \mathsf{timeout}$ then reject

else if $\exists j \neq i :\ \text{out}_{i,j} = \mathbf{msg}\langle j, \text{sign}(j, \texttt{c}), \texttt{t}'\rangle$ and $(\mid \texttt{t} - \texttt{t}' \mid > \Delta)$ then reject

else if $\exists j :\ \text{out}_{i,j} = \mathbf{msg}\langle j, \text{sign}(j, \texttt{c}), \texttt{t}'\rangle$ and not $\mathit{valid}(\text{sign}(\texttt{P}_j, \texttt{c}))$ then reject

else accept

Fig. 1. The simple MPCS protocol

The basic idea of the SIGN protocol is that everybody broadcasts her own signature and receives everybody else's signature. The network annotates all incoming send requests with the name of the sender and the time of send, and seals the resulting message. It also hands a copy of the actual network message to the sender. The protocol result for $\texttt{P}_i$ is out_i, which is the sequence of signatures or timeouts of all others, plus either her own proof-of-send or an abort. A timeout occurs when more than Δ time units have passed since broadcasting her own signature. If all goes well, every participant has in the end a list of messages containing all signatures, and a copy of her own broadcasted message to prove her commitment. This will be her signed contract.

An honest participant will first broadcast, then start receiving, and will decide in the beginning whether her personal decision is abort or not. A misbehaving participant can receive messages before broadcasting, can change her mind at any time, and can declare that a timeout happened even when this is not the case. The protocol ensures that the misbehavior is not harmful, or only harmful for the dishonest participants themselves.

In the second subprotocol, VERIFY , a presented contract (the message array out_i) is accepted only when all the messages included are timestamped within Δ time units from i's proof-of-send (before or after) and all signatures included are correct. The verifier does not need interaction with the participants when evaluating a contract. We assume he has means to verify the digital signatures of the participants. Note that some dishonest participants who broadcast only after receiving could still end up with a valid contract. This does not violate the fairness, though.

The *fairness* of the protocol is guaranteed basically by the symmetry of the checks that `Verifier` performs. *Timeliness* is ensured by the bounded delay, the timeouts and the finite number of messages. Note that this protocol is not *abuse-free*. Also note the following possible attack: if only a subset of $\{\mathtt{P}_1 \cdots \mathtt{P}_N\}$ send their signatures, a dishonest party can prove to the verifier that he has a contract with that subset of participants. This can be easily prevented by explicitly naming $\{\mathtt{P}_1 \cdots \mathtt{P}_N\}$ in the body of the contract `c`.

4 Formal correctness proof

In this section we formalize the MPCS protocol of the previous section and prove its fairness for any number of participants. We use algebraic manipulation in μCRL [11], an extension of the process algebra ACP with abstract data types, to achieve this.

From the process algebraic point of view, everything is a process, or more precisely, the behavior of every system can be modeled as process. The modelling can take place at various abstraction levels. Typically, two descriptions are distinguished: a *specification Spec* capturing the global desired behavior of the system, and the *implementation Imp*, capturing as much as possible from the implementation details of the system. The typical process algebraic correctness statement is then $Spec = \tau_I Imp$, meaning that the specification is behaviourally equivalent to the implementation, after abstraction of internal communications in I. In μCRL, processes are built from atomic actions from a set *Act* and operators for sequential, non-deterministic and parallel composition. For any processes p and q, $p + q$ denotes non-deterministic choice between p and q, $p \cdot q$ denotes their sequential composition, and $p \| q$ denotes the parallel composition (defined in terms of interleaving and synchronous communication). Synchronization is governed by a communication function γ. There is also an *encapsulation* operator ∂_H, that forces processes to communicate, by making the actions in H act exclusively in communication. The *hiding* operator τ_I abstracts away the actions in I. Finally, there are two special processes: δ (deadlock, the unit of $+$) and τ (internal action). In order to use abstract data types in a specification, a signature of multiple sorts and functions can be declared, and axiomatized by equations. Atomic actions can be parameterized with data elements, as in $\mathsf{proof}(p)$. To model input, the alternative choice such as $\sum_{x:\mathtt{NetMsg}} P(x)$ is used. Finally, if b is a term of data domain *Bool* and p and q are processes, then the conditional $(p \triangleleft b \triangleright q)$ is the process "p if b, else q". There is a powerful toolset that supports μCRL [3].

Protocol Formalization

All the entities involved in the protocol are processes: the participants $\mathtt{P}_i$, the verifier `Verifier`, and the communication medium `Net`. They are specified in Fig. 2. We do not make a distinction between honest and dishonest participants at the level of process description. In the trace set of every process $\mathtt{P}_i$ there are both well-behaved and bad traces. An honest participant will execute a well-behaved trace, for instance $\mathsf{bcast}(..).\mathsf{proof}(..).\mathsf{deliver}(..).\mathsf{claim}(..)$.

The processes communicate by synchronizing on certain actions. Here are the pairs of actions, together with the result of their communication:

$$\begin{array}{llll} \gamma\,(\mathsf{bcast},\mathsf{Bcast}) = \mathsf{BCAST} & & \gamma\,(\mathsf{proof},\mathsf{Proof}) & = \mathsf{PROOF} \\ \gamma\,(\mathsf{claim},\mathsf{Claim}) = \mathsf{CLAIM} & & \gamma\,(\mathsf{deliver},\mathsf{Deliver}) & = \mathsf{DELIVER} \end{array}$$

Because μCRL cannot model time naturally, some choices and simplifications are made. The most important is that we do not model explicitly the part where a participant is waiting for the Δ time units to pass while collecting messages, but instead we compress it in one communication step to the network ($\mathsf{DELIVER}$). It is future work to better accommodate the timestamps, possibly using the technique proposed in [4].

B, M are lists of network messages, indexed by their sender. We make use of a function $sender(x)$ which returns the first element of the tuple $x \equiv \mathbf{msg}\langle i, m, \mathtt{t}\rangle$. B_i and M_i denote the network message on the position i in the respective lists. The μCRL syntax does not allow to pass other processes as parameters. However, in order not to complicate matters with extra datatypes, we abuse the notation and write $\mathtt{P}_i(\{\mathtt{P}_1 \cdots \mathtt{P}_N\})$ for the behavior of $\mathtt{P}_i$ when participating in a protocol with $\{\mathtt{P}_1 \cdots \mathtt{P}_N\}$.

The proof

The fairness property states that any honest participant must be able to prove to the verifier that the contract has been signed, or else no other participant should be able to do that. Without loss of generality, let $\mathtt{P}_1$ be an honest participant and $\mathtt{P}_2$ be one of the other N participants, honest or not. We have to prove that either the claim of $\mathtt{P}_1$ is accepted, or, if the claim of $\mathtt{P}_1$ is rejected, the claim of $\mathtt{P}_2$ should be rejected as well. We can express this requirement as a system equivalence by hiding all actions in the system except the relevant ones (viz. $\mathsf{accept}(1)$, $\mathsf{accept}(2)$, $\mathsf{reject}(1)$, $\mathsf{reject}(2)$) and request that what's left is equivalent to a fair behavior:

$$(\forall N)\ \mathtt{CSign}'(\{\mathtt{P}_1 \cdots \mathtt{P}_N\}) \sim \mathtt{FairSystem}'$$

where

$$\mathtt{CSign}'(\{\mathtt{P}_1, \mathtt{P}_2, \cdots, \mathtt{P}_N\}) \stackrel{\mathtt{not}}{=} \tau_{\{\mathsf{accept}(i),\mathsf{reject}(i) \mid i>2\}}\mathtt{CSign}(\{\mathtt{P}_1 \cdots \mathtt{P}_N\})$$

$$\mathtt{FairSystem}' \stackrel{\mathtt{def}}{=} \mathsf{accept}(1) \| (\mathsf{accept}(2) + \mathsf{reject}(2)) + \mathsf{reject}(1) \| \mathsf{reject}(2)$$

$$
\begin{array}{lll}
\mathtt{P}_i(\{\mathtt{P}_1 \cdots \mathtt{P}_N\}) & = & \mathsf{bcast}(\mathrm{sign}(i, \mathsf{c})). \\
 & & \sum_{p:\mathtt{NetMsg}} \mathsf{proof}(p). \\
 & & \sum_{M:(\forall m \in M \; |tstp(p) - tstp(m)| < \Delta \wedge sender(m) \in \{\mathtt{P}_1 \cdots \mathtt{P}_N\})} \\
 & & \qquad \mathsf{deliver}(M). \\
 & & \mathsf{claim}(M[M_i := p]) \\
 & & \; + \\
 & & \sum_{M:\mathtt{NetMsgList}} \mathsf{deliver}(M). \\
 & & \mathsf{claim}(M[M_i := \mathsf{abort}]) \\
 & & \\
\mathtt{Net}(B) & = & \sum_x \mathsf{Bcast}(x).\mathsf{Proof}(msg(x)).\mathtt{Net}(B + msg(x)) \\
 & & \; + \\
 & & \mathsf{timeout}.\mathtt{NetDel}(B) \\
\mathtt{NetDel}(B) & = & \mathsf{Deliver}(M).\mathtt{NetDel}(B) \lhd M \subseteq B \rhd \delta \\
 & & \\
\mathtt{Verifier} & = & \sum_M \mathsf{Claim}(i, M). \\
 & & (\mathsf{accept}(i) \lhd check(i, M) \rhd \mathsf{reject}(i)) \\
 & & \\
\mathtt{CSign}(\{\mathtt{P}_1 \cdots \mathtt{P}_N\}) & = & \tau_{\{\mathsf{BCAST}, \mathsf{PROOF}, \mathsf{DELIVER}, \mathsf{CLAIM}\}} \\
 & & \partial_{\{\mathsf{bcast}, \mathsf{Bcast}, \mathsf{proof}, \mathsf{Proof}, \mathsf{claim}, \mathsf{Claim}, \mathsf{deliver}, \mathsf{Deliver}\}} \\
 & & \qquad (\; \mathtt{Net}(\emptyset) \parallel \mathtt{Verifier} \parallel \mathtt{P}_1 \parallel \cdots \parallel \mathtt{P}_N \;)
\end{array}
$$

Fig. 2. The μCRL description

and $\sim$ is weak bisimulation equivalence. To prove this equivalence, we first need a more general compositionality result for μCRL processes.

Lemma 4.1 *Let $\sim$ be a congruence for all operators of μCRL. For any process P, let $\mathtt{com}(()P)$ denote the set of actions of P used for synchronization with other processes. Let S, P, Q, R be any processes with $A = \mathtt{com}(S)$, $H = \mathtt{com}(P) \cup \mathtt{com}(Q) \cup \mathtt{com}(R)$ and $I = \gamma(A, H)$. Then, if $\gamma(\mathtt{com}(P), \mathtt{com}(Q)) = \emptyset$, $\gamma(\mathtt{com}(R), \mathtt{com}(Q)) = \emptyset$, $\gamma(\mathtt{com}(P), \mathtt{com}(R)) = \emptyset$, the following implication holds:*

$$\textit{if } \tau_I \partial_H(S||P) \sim \tau_I \partial_H(S||Q) \textit{ then } \tau_I \partial_H(S||P||R) \sim \tau_I \partial_H(S||Q||R).$$

Proof. We rely on some of the conditional alphabet axioms from [13], namely:

$$
\begin{array}{lll}
\text{(CA1)} & \gamma(\alpha(x), \alpha(y) \cap H) \subseteq H & \rightarrow \partial_H(x \| y) = \partial_H(x \| \partial_H(y)) \\
\text{(CA2)} & \gamma(\alpha(x), \alpha(y) \cap I)) \subseteq \emptyset & \rightarrow \tau_I(x \| y) = \tau_I(x \| \tau_I(y)) \\
\text{(CA3)} & \alpha(x) \cap H = \emptyset & \rightarrow \partial_H(x) = x \\
\text{(CA4)} & \alpha(x) \cap I = \emptyset & \rightarrow \tau_I(x) = x \\
\text{(CA7)} & H \cap I = \emptyset & \rightarrow \tau_I \partial_H(x) = \partial_H \tau_I(x)
\end{array}
$$

Since $\tau_I \partial_H(S||P) \sim \tau_I \partial_H(S||Q)$ and since $\sim$ is a congruence w.r.t. $||$, we can write

$$\tau_I \partial_H(S||P)||R \sim \tau_I \partial_H(S||Q)||R. \quad (1)$$

R doesn't perform any actions from I and $\tau_I \partial_H(S||P)$ doesn't either. Therefore, by axioms $(CA2)$ and $(CA4)$, the left side of (1) transforms as follows:

$$\tau_I \partial_H(S||P)||R \overset{CA4}{=} \tau_I(\tau_I \partial_H(S||P)||R) \overset{CA2}{=} \tau_I(\partial_H(S||P)||R).$$

We apply the same transformation in the right-hand side:

$$\tau_I(\partial_H(S||P)||R) \sim \tau_I(\partial_H(S||Q)||R).$$

Then, since $\sim$ is a congruence w.r.t. ∂_H, we further obtain

$$\partial_H \tau_I(\partial_H(S||P)||R) \sim \partial_H \tau_I(\partial_H(S||Q)||R).$$

We have $I \cap H = \emptyset$, thus from $(CA7)$: $\tau_I \partial_H(\partial_H(S||P)||R) \sim \tau_I \partial_H(\partial_H(S||Q)||R)$. Next, note that $\alpha(R) \subseteq H \cup Ext(R)$ and $\alpha(S||P) \subseteq A \cup H \cup I \cup Ext(S) \cup Ext(P)$.
Thus $\gamma(\alpha(R), \alpha(S||P) \cap H) = \emptyset \subseteq H$. With $(CA1)$, this means

$$\tau_I \partial_H(\partial_H(S||P)||R) = \tau_I \partial_H(S||P||R).$$

Finally, after a similar transformation on the right side, we conclude that

$$\tau_I \partial_H(S||P||R) \sim \tau_I \partial_H(S||Q||R).$$

□

We are now ready for the fairness theorem.

Theorem 4.2 $(\forall N)$ $\texttt{CSign}'(\{\texttt{P}_1, \texttt{P}_2, \cdots, \texttt{P}_N\}) \sim \texttt{FairSystem}'$.

Proof idea. Weak bisimulation is not a congruence with respect to $+$ and therefore, in order to be able to use Lemma 4.1, we will prove a stronger but less intuitive equation, namely:

$$(\forall N)\ \texttt{CSign}'(\{\texttt{P}_1, \texttt{P}_2, \cdots, \texttt{P}_N\}) \sim_{BB} \texttt{FairSystem}'_{BB} \quad (2)$$

with $\sim_{BB}$ being the rooted branching bisimulation congruence and

$$\texttt{FairSystem}^{BB} \overset{\mathsf{def}}{=} \tau.\mathsf{accept}(1) \| (\tau.\mathsf{accept}(2) + \tau.\mathsf{reject}(2)) + \tau.\mathsf{reject}(1) \| \tau.\mathsf{reject}(2)$$

(The silent prefixes are needed for the rooted branching bisimulation relation.) The proof proceeds along the following line:

(i) It holds that $\texttt{FairSystem}'_{BB} \sim_{BB} \texttt{CSign}'(\{\texttt{P}_1, \texttt{P}_2\})$. This can be shown by automatic equivalence checking. The two sides of the equation are μCRL processes with finite (and small) state spaces. The μCRL toolset can then just perform a mechanical check that the two state spaces are indeed equivalent under the equivalence notion considered, namely rooted branching bisimulation.

(ii) It holds that $\forall i\ \mathtt{CSign}'(\{\mathtt{P}_1,\mathtt{P}_2\}) \sim_{BB} \mathtt{CSign}'(\{\mathtt{P}_1,\mathtt{P}_2,\mathtt{P}_i\})$. Similarly, because the processes involved are finite-state (and in fact, small), this equivalence can be checked by the toolset.

(iii) Next follows the generalization step, where we do not rely on the toolset anymore, but on equational reasoning. Suppose

$$\mathtt{CSign}'(\{\mathtt{P}_1,\mathtt{P}_2\}) \sim_{BB} \mathtt{CSign}'(\mathtt{P}_1,\mathtt{P}_2,\cdots,\mathtt{P}_{i-1}\}).$$

By Lemma 4.1 it follows that

$$\mathtt{CSign}'(\{\mathtt{P}_1,\mathtt{P}_2,\mathtt{P}_i\}) \sim_{BB} \mathtt{CSign}'(\mathtt{P}_1,\mathtt{P}_2,\cdots,\mathtt{P}_{i-1},\mathtt{P}_i\})$$

and, by point (ii),

$$\mathtt{CSign}'(\{\mathtt{P}_1,\mathtt{P}_2\}) \sim_{BB} \mathtt{CSign}'(\mathtt{P}_1,\mathtt{P}_2,\cdots,\mathtt{P}_{i-1},\mathtt{P}_i\}).$$

We thus arrive at the equation

$$\mathtt{CSign}'(\{\mathtt{P}_1,\mathtt{P}_2\}) \sim_{BB} \mathtt{CSign}'(\mathtt{P}_1,\mathtt{P}_2,\cdots,\mathtt{P}_N\}),$$

saying that all the messages exchanged in the network do not affect the observable activity of participants $\mathtt{P}_1$ and $\mathtt{P}_2$. From point (i), the targeted result follows.

5 Discussion on the absence and presence of a TTP

Above, we have shown that a reliable network model can function in place of a trusted referee in a multiparty contract signing protocol. Now, we discuss this result from several perspectives. First, we point out the differences in functionality between a TTP as used traditionally and the reliable network used here. We argue that it is difficult to relax the proposed model. Next, we present in a unified framework three problems closely related to contract signing. We speculate on a possible solution of the problems assuming a reliable network as deployed above and formulate some related challenges.

A TTP vs. a reliable network

Usually [5,9,2], a TTP plays an active role in a protocol: it checks signatures, it is aware of the participants engaged, sometimes it maintains a history of interactions with the participants, etc. These operations need memory, processing power and a dedicated contract signing mediation algorithm. When a coordination architecture is used for the this purpose, we may expect that the communication is made reliable by transparent network algorithms. There may be support from sophisticated interaction mechanisms in the middleware. There may be even data storage facilities. But we cannot assume computing power available in the communication network. In particular, no awareness of the existence and activity of the protocol's participants can be assumed.

Weaker communication models?

In our search for a TTP-free contract signing protocol in a communication model that is as weak as possible, it remains unsettled what the weakest model is. We therefore have a closer look at the model of Section 3 and see whether its requirements can be relaxed.

The *reliable broadcast* primitive can be replaced by reliable send. The protocol would remain fair. In fact, there will be more possibilities for a dishonest player to harm herself by doing a send without waiting for the corresponding receive. On a side note, reliable channels are not such a powerful thing to ask for. In the traditional communication model one also has to assume some reliable channels hooking up to the TTP.

The *time* elements cannot be dropped. Without the bound on the network delay there is no way to tell whether an expected message has been sent or not. With the bounded delay, we know that a timeout means that the expected message has not been sent. Timestamps are necessary in order to prevent dishonest participants from holding up their signatures too long, causing honest participants to timeout and abort, leaving the dishonest parties with a valid signed contract.

The *proofs of send* can, in principle, be replaced by proofs of delivery to obtain a more natural communication model. This would have the advantage that not all channels need to be reliable, but only those on which the acknowledgments are sent. This scheme, however, doesn't immediately fit the broadcast primitive and it will therefore be more complicated to analyze formally as well.

Related problems

There is a series of well-known problems that are related to contract signing, although, from a theoretical point of view, the relationship is not completely clear. These problems are fair exchange, distributed consensus and multiparty computation. Here, we define them formally, in a minimalistic way, i.e., stripped of all unessential information like signatures, secret/public key cryptography, item descriptions etc. The participants that behave as expected are called honest, the others dishonest — regardless of whether their misbehavior is due to bad intentions or to failures. Let t be the maximum number of dishonest players (a bound a priori known). All the input values are private to the respective participants. To solve the problem means to find a terminating algorithm with the specified input/output behavior.

- *Multiparty contract signing* (MPCS)

Participants	$\mathtt{P}_1 \cdots \mathtt{P}_N, \mathtt{Verifier}$
Input	$\langle\, \mathrm{d}_1 \cdots \mathrm{d}_N \,\rangle$, a verification function *check*
Output	$\langle\, \mathrm{out}_1 \cdots \mathrm{out}_N \,\rangle$
Requirements	(fairness) for any honest $\mathtt{P}_i$, if $\mathit{check}(\mathrm{out}_i) = \mathtt{false}$ then $\mathit{check}(\mathrm{out}_j) = \mathtt{false}$ for all participants

`Verifier` is always honest and its role is just to apply the function *check*, which is defined on all possible outcomes.

- *Distributed consensus* (DC)

Participants	$\mathtt{P}_1 \cdots \mathtt{P}_N$
Input	$\langle\, \mathrm{d}_1 \cdots \mathrm{d}_N \,\rangle$
Output	$\langle\, \mathrm{out}_1 \cdots \mathrm{out}_N \,\rangle$
Requirements	(agreement) for any honest $\mathtt{P}_i, \mathtt{P}_j$, $\mathrm{out}_i = \mathrm{out}_j$ (validity) if for any honest $\mathtt{P}_i, \mathtt{P}_j$, $\mathrm{d}_i = \mathrm{d}_j$, then for any honest $\mathtt{P}_i$, $\mathrm{out}_i = \mathrm{d}_i$.

DC is known to be impossible in the crash model, if $t > N/3$ or if t exceeds the connectivity degree of the communication graph [8]. On the other hand, solutions exist for the case $t < \frac{1}{3}N$ (see [9]).

- *Multiparty computation* (MC)

Participants	$\mathtt{P}_1 \cdots \mathtt{P}_N$, multiple variable function F
Input	$\langle\, \mathrm{d}_1 \cdots \mathrm{d}_N \,\rangle$
Output	$\langle\, \mathrm{out}_1 \cdots \mathrm{out}_N \,\rangle$
Requirements	(correctness) for any honest $\mathtt{P}_i, \mathtt{P}_j$, $\mathrm{out}_i = F(\mathrm{d}_1 \cdots \mathrm{d}_N)$ (privacy) d_i should remain private

MC has a probabilistic solution for $t < \frac{1}{3}N$ (see [10]).

- *Multiparty fair exchange* (MPFE):

Participants	$\mathtt{P}_1 \cdots \mathtt{P}_N$
Input	$\langle\, \mathrm{d}_1 \cdots \mathrm{d}_N \,\rangle$, $\pi : \{1 \cdots N\} \rightarrow \{1 \cdots N\}$
Output	$\langle\, \mathrm{out}_1 \cdots \mathrm{out}_N \,\rangle$
Requirements	(fairness) for any honest $\mathtt{P}_i$, $\mathrm{out}_i = \mathrm{d}_{\pi(i)}$

Here, π is a permutation indicating how the items are to be transferred; the item owned by $\mathtt{P}_\pi(i)$ is desired by and should be given to $\mathtt{P}_i$. In its most general form, the fair exchange problem allows every participant to obtain items from a set of other participants, not just one. The problem has no TTP-free solution already for the two-party case [15]. This has been proved by reduction of 2-party DC to 2-party FE.

Looking for the 'weakest' communication model in which the listed problems can be solved without TTP will help to clarify their relationship. For example, it is conjectured [15], that two-party fair exchange is more difficult to solve than consensus. To prove this, it is enough to find a communication model in which consensus is possible, but fair exchange not. The model we used in this paper might be a possible answer. Consensus is solvable in this model, but, at present, we do not have a proof that MPFE is not. We also conjecture that, because of the presence of the honest verifier, MPCS is not more difficult than DC.

6 Conclusion

Existent non-probabilistic contract signing protocols make use of a trusted third party. The main issue addressed in this paper is the transfer of trust from a dedicated trusted third party towards a generic communication medium. We proposed a reliable communication network and a basic contract signing protocol built on top of it.

Because of the strong guarantees of the underlying communication model, our protocol has limited practical use for highly unreliable media like the Internet. There, at least for the moment, the most appropriate contract signing protocols remain the optimistic ones [2,9], which rely on the presence of a trusted party ready to be contacted in case of conflicts. However, in the context of middleware platforms, the additional network requirements on which our protocol relies may very well be met.

We have formally proved that the proposed protocol preserves fairness for any number of participants. Given the simplicity of the protocol and the strong network assumptions, this will not come as a surprise. However, the proof provides a starting point for formal verification of parametrized multiparty contract signing protocols. The techniques are promising, but need to be developed further in order to deal with less abstract, more detailed protocols that are substantially more complex.

As future work, we plan to analyze other (optimistic) MPCS protocols. The Baum-Waidner protocol [2], for instance, is symmetric and optimistic, and therefore a convenient next target for the parametrized verification techniques proposed here. Taking inspiration in the solution for MC presented in [10], we would also like to explicitly define a MPCS protocol without TTP in the weaker communication model used there, and under the essential assumption $t < \frac{1}{3}N$.

On the more theoretical side, it would be interesting to investigate the relations between the problems mentioned in Section 5, by identifying the communication models where they are or are not solvable without the interference of a TTP.

References

[1] Anderson, R., "Security Engineering: A guide to building dependable systems," Wiley, 2001.

[2] Baum-Waidner, B. and M. Waidner, *Round-optimal and abuse free optimistic multi-party contract signing*, in: U. Montanari, J. Rolim and E. Welzl, editors, *ICALP*, LNCS **1853**, 2000, pp. 524–535.

[3] Blom, S., W. Fokkink, J. Groote, I. Langevelde, B. Lisser and J. Pol, *μCRL: A toolset for analysing algebraic specifications*, in: G. Berry, H. Comon and A. Finkel, editors, *Proceedings CAV'01*, LNCS **2102**, 2001, pp. 250–254.

[4] Blom, S., N. Ioustinova and N. Sidorova, *Timed verification with μCRL*, in: M. Broy and A. Zamulin, editors, *Proc. PSI'03*, LNCS **2890**, 2003, pp. 178–192.

[5] Cederquist, J. and M. Dashti, *Formal analysis of a fair payment protocol*, in: T. Dimitrakos and F. Martinelli, editors, *Proc. FAST 2004*, IFIP International Federation for Information Processing **173** (2005).

[6] Chadha, R., S. Kremer and A. Scedrov, *Formal analysis of multi-party contract signing*, Journal of Automated Reasoning (2005), to appear.

[7] Chen, L., C. Kudla and K. Paterson, *Concurrent signatures*, in: C. Cachin and J. Camenisch, editors, *Proc. EUROCRYPT*, LNCS **3027**, 2004, pp. 287–305.

[8] Fischer, M., N. Lynch and M. Merritt, *Easy impossibility proofs for distributed consensus problems*, Distributed Computing **1** (1986), pp. 26–39.

[9] Garay, J., M. Jakobsson and P. MacKenzie, *Abuse-free optimistic contract signinng*, in: M. Wiener, editor, *Proc. CRYPTO'99*, LNCS **1666**, 1999, pp. 449–466.

[10] Goldreich, O., S. Micali and A. Wigderson, *How to play any mental game*, in: *Proc. STOC '87*, 1987, pp. 218–229.

[11] Groote, J. and M. Reniers, *Algebraic process verification*, in: J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, North-Holland, 2001 pp. 1151–1208.

[12] Groote, J. and J. van Wamel, *The parallel composition of uniform processes with data*, Theoretical Computer Science **266** (2001), pp. 631–652.

[13] Korver, H. and M. Sellink, *Example verifications using alphabet axioms*, Formal Aspects of Computing **10** (1998), pp. 43–58.

[14] Object Management Group, "Common Object Request Broker Architecture: Core Specification," version 3.0.3 edition (2004).

[15] Pagnia, H. and F. C. Gärtner, *On the impossibility of fair exchange without a Trusted Third Party*, Technical Report TUD-BS-1999-02, Darmstadt University of Technology (1999).

[16] Pang, J., J. Pol and M. Valero Espada, *Abstraction of parallel uniform processes with data*, in: *Proc. SEFM'04* (2004), pp. 14–23.

[17] Sun Microsystems Inc., "JavaSpaces Specification," 1.0 beta edition (1998).

[18] Wyckoff, P., S. McLaughry, T. Lehman and D. Ford, *TSpaces*, IBM Systems Journal **37** (1998), pp. 454–474.