Generalizing Multi-party Contract Signing

Sjouke Mauw[1] and Saša Radomirović[2]

[1] CSC/SnT, University of Luxembourg
sjouke.mauw@uni.lu

[2] Institute of Information Security, Dept. of Computer Science, ETH Zurich
sasa.radomirovic@inf.ethz.ch

Abstract. Multi-party contract signing (MPCS) protocols allow a group of signers to exchange signatures on a predefined contract. Previous approaches considered either completely linear protocols or fully parallel broadcasting protocols. We introduce the new class of DAG MPCS protocols which combines parallel and linear execution and allows for parallelism even within a signer role. This generalization is useful in practical applications where the set of signers has a hierarchical structure, such as chaining of service level agreements and subcontracting.

Our novel DAG MPCS protocols are represented by directed acyclic graphs and equipped with a labeled transition system semantics. We define the notion of *abort-chaining sequences* and prove that a DAG MPCS protocol satisfies fairness if and only if it does not have an abort-chaining sequence. We exhibit several examples of optimistic fair DAG MPCS protocols. The fairness of these protocols follows from our theory and has additionally been verified with our automated tool.

We define two complexity measures for DAG MPCS protocols, related to execution time and total number of messages exchanged. We prove lower bounds for fair DAG MPCS protocols in terms of these measures.

1 Introduction

A multi-party contract signing (MPCS) protocol is a communication protocol that allows a number of parties to sign a digital contract. The need for MPCS protocols arises, for instance, in the context of service level agreements (SLAs) and in supply chain contracting. In these domains (electronic) contract negotiations and signing are still mainly bilateral. Instead of negotiating and signing one multi-party contract, in practice, multiple bilateral negotiations are conducted in parallel [20]. Because negotiations can fail, parties may end up with just a subset of the pursued bilateral contracts. If a party is missing contracts with providers or subcontractors, it faces an *overcommitment* problem. If contracts with customers are missing, it has an *overpurchasing* problem [8]. Both problems can be prevented by using fair multi-party contract signing protocols.

Existing optimistic MPCS protocols come in two flavors. *Linear* MPCS protocols require that at any point in time at most one signer has enough information to proceed in his role by sending messages to other signers. *Broadcast* MPCS protocols specify a number of communication rounds in each of which all signers

R. Focardi and A. Myers (Eds.): POST 2015, LNCS 9036, pp. 156–175, 2015.
DOI: 10.1007/978-3-662-46666-7_9

send or broadcast messages to each other. However, neither of the two kinds of protocols is suitable for SLAs or supply chain contracting. The reason is that in both domains, the set of contractors typically has a hierarchical structure, consisting of main contractors and levels of subcontractors. It is undesirable (and perhaps even infeasible) for the main contracting partners and their subcontractors to directly communicate with another partner's subcontractors. This restriction immediately excludes broadcast protocols as potential solutions and forces linear protocols to be impractically large.

In this paper we introduce MPCS protocol specifications that support arbitrary combinations of linear and parallel actions, even within a protocol role. The message flow of such protocols can be specified as a directed acyclic graph (DAG) and we therefore refer to them as *DAG* MPCS protocols.

A central requirement for MPCS protocols is *fairness*. This means that either all honest signers get all signatures on the negotiated contract or nobody gets the honest signers' signatures. It is well known that in asynchronous communication networks, a deterministic MPCS protocol requires a trusted third party (TTP) to achieve fairness [5]. *Optimistic* MPCS protocols [1] involve the TTP only when conflicts or faults occur and thus prevent the TTP from becoming a bottleneck. We focus on optimistic protocols in this paper.

DAG MPCS protocols not only allow for better solutions to the subcontracting problem, but also have further advantages over linear and broadcast MPCS protocols and we design three novel MPCS protocols that demonstrate this. One such advantage concerns communication complexity. Linear protocols can reach the minimal number of messages necessary to be exchanged in fair MPCS protocols at the cost of a high number of protocol "rounds". We call this the *parallel complexity*, which is a generalization of the round complexity measure for broadcast protocols, and define it in Section 4.3. Conversely, broadcast protocols can attain the minimal number of protocol rounds necessary for fair MPCS, but at the cost of a high message complexity. We demonstrate that DAG MPCS protocols can simultaneously attain best possible order of magnitude for both complexity measures.

As discussed in our related work section, the design of fair MPCS protocols has proven to be non-trivial and error-prone. We therefore not only prove our three novel DAG MPCS protocols to be fair, but we also derive necessary and sufficient conditions for fairness of any optimistic DAG MPCS protocol. These conditions can be implemented and verified automatically, but they are still non-trivial. Therefore, for a slightly restricted class of DAG protocols, we additionally derive a fairness criterion that is easy to verify.

Contributions. Our main contributions are (i) the definition of a syntax and interleaving semantics of DAG MPCS protocols (Section 4.1); (ii) the definition of the message complexity and parallel complexity of such protocols (Section 4.3); (iii) a method to derive a full MPCS specification from a *skeletal graph*, including the TTP logic (Section 5); (iv) necessary and sufficient conditions for fairness of DAG MPCS protocols (Section 6); (v) minimal complexity bounds for DAG

MPCS protocols (Section 7.1); (vi) novel fair MPCS protocols (Section 7.2); (vii) a software tool that verifies whether a given MPCS protocol is fair.[1]

2 Related Work

We build on the body of work that has been published in the field of fair optimistic MPCS protocols in asynchronous networks. The first such protocols were proposed by Baum-Waidner and Waidner [2], viz. a round-based broadcast protocol and a related round-based linear protocol. They showed subsequently [3] that these protocols are round-optimal. This is a complexity measure that is related to, but less general than, parallel complexity defined in the present paper.

Garay et al. [6] introduced the notion of *abuse-free* contract signing. They developed the technique of *private contract signature* and used it to create abuse-free two-party and three-party contract signing protocols. Garay and MacKenzie [7] proposed MPCS protocols which were later shown to be unfair using the model checker Mocha and improved by Chadha et al. [4]. Mukhamedov and Ryan [17] developed the notion of *abort chaining attacks* and used such attacks to show that Chadha et al.'s improved version does not satisfy fairness in cases where there are more than five signers. They introduced a new optimistic MPCS protocol and proved fairness for their protocol by hand and used the NuSMV model checker to verify the case of five signers. Zhang et al. [21] have used Mocha to analyze the protocols of Mukhamedov and Ryan and of Mauw et al. [15].

Mauw et al. [15] used the notion of abort chaining to establish a lower bound on the message complexity of linear fair MPCS protocols. This complexity measure is generalized in the present paper to DAG MPCS protocols. Kordy and Radomirović [10] have shown an explicit construction for fair linear MPCS protocols. The construction covers in particular the protocols proposed by Mukhamedov and Ryan [17] and the linear protocol of Baum-Waidner and Waidner [2], but not the broadcast protocols. The DAG MPCS protocol model and fairness results developed in the present paper encompass both types of protocols. MPCS protocols combining linear and parallel behaviour have not been studied yet.

Apart from new theoretical insights to be gained from designing and studying DAG MPCS protocols, we anticipate interesting application domains in which multiple parties establish a number of related contracts, such as SLAs. Emerging business models like Software as a Service require a negotiation to balance a customer's requirements against a service provider's capabilities. The result of such a negotiation is often complicated by the dependencies between several contracts [13] and multi-party protocols may serve to mitigate this problem. Karaenke and Kirn [8] propose a multi-tier negotiation protocol to mitigate the problems of overcommitment and overpurchasing. They formally verify that the protocol solves the two observed problems, but do not consider the fairness problem. SLAs and negotiation protocols have also been studied in the multi-agent community. An example is the work of Kraus [11] who defines a multi-party

[1] Proofs of theorems and additional results including a description of the tool and a link to it are given in the extended version of this paper [14].

negotiation protocol in which agreement is reached if all agents accept an offer. If the offer is rejected by at least one agent, a new offer will be negotiated.

Another interesting application area concerns *supply chain contracting* [12]. A supply chain consists of a series of firms involved in the production of a product or service with potentially complex contractual relationships. Most literature in this area focuses on economic aspects, like pricing strategies. An exception is the recent work of Pavlov and Katok [9] in which fairness is studied from a game-theoretic point of view. The study of multi-party signing protocols and multi-contract protocols has only recently been identified as an interesting research topic in this application area [19].

3 Preliminaries

3.1 Multi-party Contract Signing

The purpose of a multi-party contract signing protocol is to allow a number of parties to sign a digital contract in a fair way. In this section we recall the basic notions pertaining to MPCS protocols. We use A to denote the set of signers involved in a protocol, $\mathfrak{C}$ to denote the contract, and T to denote the TTP.

A signer is considered *honest* (cf. Definition 5) if it faithfully executes the protocol specification. An MPCS protocol is said to be *optimistic* if its execution in absence of adversarial behaviour and failures and with all honest signers results in signed contracts for all participants without any involvement of T. Optimistic MPCS protocols consist of two subprotocols: the *main* protocol that specifies the exchange of *promises* and *signatures* by the signers, and the *resolve* protocol that describes the interaction between the agents and T in case of a failure in the main protocol. A promise made by a signer indicates the intent to sign $\mathfrak{C}$. A promise $\wp_P(m, x, Q, \mathrm{T})$ can only be generated by signer $P \in A$. The content (m, x) can be extracted from the promise and the promise can be verified by signer $Q \in A$ and by T. A signature $\mathcal{S}_P(m)$ can only be generated by P and by T, if T has a promise $\wp_P(m, x, Q, \mathrm{T})$. The content m can be extracted and the signature can be verified by anybody. Cryptographic schemes that allow for the above properties are digital signature schemes and private contract signatures [6].

MPCS protocols must satisfy at least two security requirements, namely *fairness* and *timeliness*. An optimistic MPCS protocol for contract $\mathfrak{C}$ is said to be *fair* for an honest signer P if whenever some signer $Q \neq P$ obtains a signature on $\mathfrak{C}$ from P, then P can obtain a signature on $\mathfrak{C}$ from all signers participating in the protocol. An optimistic MPCS protocol is said to satisfy *timeliness*, if each signer has a recourse to stop endless waiting for expected messages. The fairness requirement will be the guiding principle for our investigations and timeliness will be implied by the communication model together with the behaviour of the TTP. A formal definition of fairness is given in Section 6.

3.2 Graphs

Let $G = (V, E)$ with $E \subseteq V \times V$ be a directed acyclic graph. Let $v, w \in V$ be vertices. We say that v *causally precedes* w, denoted $v \prec w$, if there is a directed path from v to w in the graph. We write $v \preceq w$ for $v \prec w \vee v = w$. We extend *causal precedence* to the set $V \cup E$ as follows. Given two edges $(v, w), (v', w') \in E$, we say that (v, w) *causally precedes* (v', w') and write $(v, w) \prec (v', w')$, if $w \preceq v'$. Similarly, we write $v \prec (v', w')$ if $v \preceq v'$ and $(v, w) \prec v'$ if $w \preceq v'$. Let $x, y \in V \cup E$. If x causally precedes y we also say that y *causally follows* x. We say that a set $S \subseteq V \cup E$ is *causally closed* if it contains all causally preceding vertices and edges of its elements, i.e., $\forall x \in S, y \in V \cup E : y \prec x \implies y \in S$.

By $\text{in}(v) \subseteq E$ we denote the set of edges incoming to v and by $\text{out}(v) \subseteq E$ the set of edges outgoing from v. Formally, we have $\text{in}(v) = \{(w, v) \in E \mid w \in V\}$ and $\text{out}(v) = \{(v, w) \in E \mid w \in V\}$.

3.3 Assumptions

The communication between signers is asynchronous and messages can get lost or be delayed arbitrary long. The communication channels between signers and the TTP T are assumed to be *resilient.* In order to simplify our reasoning, we assume that the channels between protocol participants are confidential and authentic. We consider the problem of delivering confidential and authentic messages in a Dolev-Yao intruder model to be orthogonal to the present problem setting.

We assume that $\mathfrak{C}$ contains the contract text along with fresh values (contributed by every signer) which prevent different protocol executions from generating interchangeable protocol messages. Furthermore we assume that $\mathfrak{C}$ contains all information that T needs in order to reach a decision regarding the contract in case it is contacted by a signer. This information contains the protocol specification, an identifier for T, identifiers for the signers involved in the protocol, and the assignment of signers to protocol roles in the protocol specification.

We assume the existence of a designated resolution process per signer which coordinates the various resolution requests of the signer's parallel threads. It will ensure that T is contacted at most once by the signer. After having received the first request from one of the signer's threads, this resolution process will contact T on behalf of the signer and store T's reply. This reply will be forwarded to all of the signer's threads whenever they request resolution.

4 DAG Protocols

Our DAG protocol model is a multi-party protocol model in an asynchronous network with a TTP and an adversary that controls a subset of parties.

4.1 Specification and Execution Model

A *DAG protocol specification* (or simply, a *protocol specification*) is a directed acyclic graph in which the vertices represent the state of a signer and the edges represent either a causal dependency between two states (an ε-edge) or the sending of a message. A vertex' outgoing edges can be executed in parallel. Edges labeled with *exit* denote that a signer contacts T.

Definition 1. *Let R be a set of roles such that $\mathrm{T} \notin R$ and M a set of messages. Let ε and exit be two symbols, such that $\varepsilon, \mathit{exit} \notin M$. By $M_\varepsilon^{\mathit{exit}}$ and R_T we denote the sets $M_\varepsilon^{\mathit{exit}} = M \cup \{\varepsilon, \mathit{exit}\}$ and $R_\mathrm{T} = R \cup \{\mathrm{T}\}$, respectively. A* DAG protocol specification *is a labeled directed acyclic graph $\mathcal{P} = (V, E, r, \mu, \delta)$, where*

1. *(V, E) is a directed acyclic graph;*
2. *$r \colon V \to R_\mathrm{T}$ is a labeling function assigning roles to vertices;*
3. *$\mu \colon E \to M_\varepsilon^{\mathit{exit}}$ is an edge-labeling function that satisfies*
   (a) $\forall (v, v') \in E \colon \mu(v, v') = \varepsilon \implies r(v) = r(v')$,
   (b) $\forall (v, v') \in E \colon \mu(v, v') = \mathit{exit} \implies r(v') = \mathrm{T}$;
4. *$\delta \colon M^* \to M$ is a function associated with exit-labeled edges.*

A message edge (v, v') specifies that $\mu(v, v') = m$ is to be sent from role $r(v)$ to role $r(v')$. An ε-edge (v, v') represents internal progress of role $r(v) = r(v')$ and allows to specify a causal order in the role's events. An exit edge denotes that a role can contact the TTP. The TTP then uses the function δ to determine a reply to the requesting role, based on the sequence of messages that it has received. In Section 5 exit messages and the δ function are used to model the resolve protocol of the TTP.

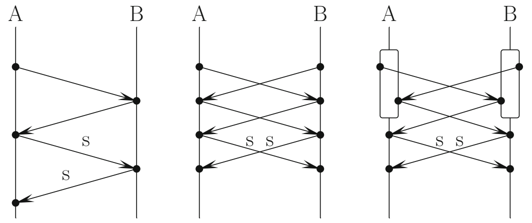


Fig. 1. Linear, broadcast, and the novel DAG MPCS protocols

We give three examples of DAG protocols in Figure 1, represented as Message Sequence Charts (MSCs). The dots denote the vertices, which we group vertically below their corresponding role names. Vertical lines in the MSCs correspond to ε-edges and horizontal or diagonal edges represent message edges. We mark edges labeled with signing messages with an "s" and we leave out the edge labels of promise messages. We do not display exit edges, they are implied by the MPCS protocol specification. A box represents the splitting of a role into two parallel threads, which join again at the end of the box. We revert to a traditional representation of labeled DAGs if it is more convenient (see, e.g., Figure 2).

The first protocol in Figure 1 is a classical linear 2-party contract signing protocol. It consists of one round of promises followed by a round of exchanging signatures. The second protocol is the classical broadcast protocol for two signers. It consists of two rounds of promises, followed by one round of signatures. The third protocol is a novel DAG protocol, showing the power of in-role parallelism. It is derived from the broadcasting protocol by observing that its fairness does not depend on the causal order of the first two vertices of each of the roles.

Let $\mathcal{P} = (V, E, r, \mu, \delta)$ be a protocol specification. The *restriction* of $\mathcal{P}$ to role P, denoted by $\mathcal{P}_P$, is the protocol specification $(V_P, E_P, r_P, \mu_P, \delta_P)$, where

$$E_P = \{(v, v') \in E \mid r(v) = P \vee r(v') = P\}\,, \quad V_P = \{v, v' \in V \mid (v, v') \in E_P\}\,,$$
$$r_P(v) = r(v) \text{ for } v \in V_P, \quad \mu_P(e) = \mu(e) \text{ for } e \in E_P, \quad \text{and} \quad \delta_P = \delta.$$

The execution state of a protocol consists of the set of events, connected to vertices or edges, that have been executed.

Definition 2. *Let $\mathcal{P} = (V, E, r, \mu, \delta)$ be a protocol specification. A* state *of $\mathcal{P}$ is a set $s \subseteq V \cup E$. The set of states of $\mathcal{P}$ is denoted by $\mathcal{S}_\mathcal{P}$. The* initial state *of $\mathcal{P}$ is defined as $s_0 = \emptyset$.*

In order to give DAG protocols a semantics, we first define the *transition relation* between states of a protocol.

Definition 3. *Let $\mathcal{P} = (V, E, r, \mu, \delta)$ be a protocol specification, $L = \{\varepsilon, send, recv, exit\}$ the set of transition labels, and $s, s' \in \mathcal{S}_\mathcal{P}$ the states of $\mathcal{P}$. We say that $\mathcal{P}$ transitions with label α from state s into s', denoted by $s \overset{\alpha}{\rightsquigarrow} s'$, iff $s \neq s'$ and one of the following conditions holds*

1. *$\alpha = recv$ and $\exists v \in V$, such that $\mathrm{in}(v) \subseteq s$ and $s' = s \cup \{v\}$,*
2. *$\alpha = send$ and $\exists m \in M, e \in E$, such that $\mu(e) = m$, and $s' = s \cup \{e\}$,*
3. *$\alpha = \varepsilon$ and $\exists e = (v, v') \in E$, such that $\mu(e) = \varepsilon$, $v \in s$ and $s' = s \cup \{e\}$,*
4. *$\alpha = exit$ and $\exists e \in E$, such that $\mu(e) = exit$ and $s' = s \cup \{e\}$.*

In Definition 3, receive events are represented by vertices, all other events by edges. By the first condition, a receive event can only occur if all events assigned to the incoming edges have occurred. In contrast, the sending of messages (second condition) can take place at any time. The third condition states that an ε-edge can be executed if the event on which it causally depends has been executed. Finally, like send events, an exit event can occur at any time. Every event may occur at most once, however. This is ensured by the condition $s' \neq s$.

The transitions model all possible behavior of the system. The behavior of honest agents in the system will be restricted as detailed in the following subsection. We denote sequences by $[a_0, a_1, \ldots, a_l]$ and the concatenation of two sequences σ_1, σ_2 by $\sigma_1 \cdot \sigma_2$.

Definition 4. *Let $\mathcal{P} = (V, E, r, \mu, \delta)$ be a protocol specification and $L = \{\varepsilon, send, recv, exit\}$ a set of labels. The* semantics *of $\mathcal{P}$ is the labeled transition system $(\mathcal{S}_\mathcal{P}, L, \rightsquigarrow, s_0)$, which is a graph consisting of vertices $\mathcal{S}_\mathcal{P}$ and edges $\rightsquigarrow$ with start state s_0. An* execution *of $\mathcal{P}$ is a finite sequence $\rho = [s_0, \alpha_1, s_1, \ldots, \alpha_l, s_l]$, $l \geq 0$, such that $\forall i \in \{0, \ldots, l-1\}$: $s_i \overset{\alpha_{i+1}}{\rightsquigarrow} s_{i+1}$. The set of all executions of $\mathcal{P}$ is denoted by* $\mathrm{Exe}(\mathcal{P})$.

If $\rho = [s_0, \alpha_1, s_1, \ldots, \alpha_l, s_l]$ is an execution of $\mathcal{P}$ and $\mathcal{P}_P$ is the restriction to role P, then the *restricted* execution ρ_P is obtained inductively as follows.

1. $[s]_P = [s \cap (V_P \cup E_P)]$ for a state s.

2. $([s, \alpha, s'] \cdot \sigma)_P = \begin{cases} [s]_P \cdot \sigma_P & \text{if } [s]_P = [s']_P \\ [s]_P \cdot [\alpha] \cdot ([s'] \cdot \sigma)_P & \text{else.} \end{cases}$

Commutativity of restriction and execution is asserted by the following lemma.

Lemma 1. *Let $\mathcal{P}$ be a protocol specification and $\mathcal{P}_P$ the restriction to role P. Then every restricted execution ρ_P is an execution of $\mathcal{P}_P$.*

4.2 Adversary Model

An honest agent executes the protocol specification faithfully. The following definition specifies what this entails for a DAG protocol: the agent waits for the reception of all causally preceding messages before sending causally following messages, does not execute an *exit* edge attached to a vertex v if all messages at v have been received and never executes more than one *exit* edge (which in the context of MPCS protocols corresponds to contacting the TTP at most once), and does not send any messages which causally follow a vertex from which the *exit* edge was executed.

Definition 5. *Let $\mathcal{P}$ be a DAG protocol specification. An agent P is* honest *in an execution ρ of $\mathcal{P}$, if all states s of the restricted execution ρ_P satisfy the following conditions:*

1. *s contains at most one exit edge.*
2. *If s contains no exit edge, then s is causally closed.*
3. *If s contains an exit edge $e = (v, w)$, $\mu(e) = exit$, then $v \notin s$ and $s \setminus \{e\}$ is causally closed.*

A dishonest agent is only limited by the execution model. Thus a dishonest agent can send its messages at any time and in any order, regardless of the causal precedence given in the protocol specification. A dishonest agent can execute multiple *exit* edges and may send and receive messages causally following an exit edge. Dishonest agents are controlled by a single adversary, their knowledge is shared with the adversary. The adversary can delay or block messages sent from one agent to another, but the adversary cannot prevent messages between agents and the TTP from being delivered eventually. All communication channels are authentic and confidential.

4.3 Communication Complexity

To define measures for expressing the communication complexity of DAG protocols, we introduce the notion of *closed executions*. A closed execution is a complete execution of the protocol by honest agents.

Definition 6. *Let $\mathcal{P} = (V, E, r, \mu, \delta)$ be a protocol specification and $(\mathcal{S}_\mathcal{P}, L, \rightsquigarrow, s_0)$ be the semantics for $\mathcal{P}$. Given $\rho = [s_0, \alpha_1, s_1, \ldots, \alpha_l, s_l] \in \mathrm{Exe}(\mathcal{P})$, we say that ρ is* closed *if the following three conditions are satisfied*

1. *s_i is causally closed, for every $0 \leq i \leq l$,*
2. *$\alpha_i \neq exit$, for every $1 \leq i \leq l$,*
3. *$\nexists \alpha \in L \setminus \{exit\}, s \in \mathcal{S}_\mathcal{P} \colon \rho \cdot [\alpha, s] \in \mathrm{Exe}(\mathcal{P})$.*

The set of all closed executions of $\mathcal{P}$ is denoted by $\mathrm{Exec_C}(\mathcal{P})$.

Let $\rho = [s_0, \alpha_1, s_1, \ldots, \alpha_l, s_l]$ be an execution of a protocol $\mathcal{P}$. By $|\rho|_{send}$ we denote the number of labels α_i, for $1 \leq i \leq l$, such that $\alpha_i = send$.

Lemma 2. *For any two closed executions ρ and ρ' of a protocol $\mathcal{P}$ we have $|\rho|_{send} = |\rho'|_{send}$.*

The proof is given in [14]. The first measure expressing the complexity of a protocol $\mathcal{P}$ is called *message complexity*. It counts the overall number of messages that have been sent in a closed execution of a protocol $\mathcal{P}$.

Definition 7. *Let $\mathcal{P}$ be a protocol specification and let $\rho \in \mathrm{Exec_C}(\mathcal{P})$. The* message complexity *of $\mathcal{P}$, denoted by $MC_\mathcal{P}$, is defined as $MC_\mathcal{P} = |\rho|_{send}$.*

Lemma 2 guarantees that the message complexity of a protocol is well defined.

The second complexity measure is called *parallel complexity*. It represents the minimal time of a closed execution assuming that all events which can be executed in parallel are executed in parallel. The parallel complexity of a protocol is defined as the length of a maximal chain of causally related send edges.

Definition 8. *The* parallel complexity *of a protocol $\mathcal{P}$, denoted by $PC_\mathcal{P}$, is defined as*

$$PC_\mathcal{P} = \max_{n \in \mathbb{N}} \exists_{[e_1, e_2, \ldots, e_n] \in E^*} : \forall_{1 \leq i \leq n} : \mu(e_i) = send \wedge \forall_{1 \leq i < n} : e_i \prec e_{i+1}.$$

Example 1. The message complexity of the first protocol in Figure 1 is 4, which is known to be optimal for two signers [18]. Its parallel complexity is 4, too. The message complexity of the other two protocols is 6, but their parallel complexity is 3, which is optimal for broadcasting protocols with two signers [3].

5 DAG MPCS Protocols

We now define a class of optimistic MPCS protocols in the DAG protocol model.

5.1 Main Protocol

The key requirements we want our DAG MPCS protocol specification to satisfy, stated formally in Definition 9, are as follows. The messages exchanged between signers in the protocol are of two types, promises, denoted by $\wp()$, and signatures, denoted by $\mathcal{S}()$. Every promise contains information about the vertex from which it is sent. This is done by concatenating the contract $\mathfrak{C}$ with the vertex v the promise originates from and is denoted by $(\mathfrak{C}, v)$. The signers can contact the TTP at any time. This is modeled with exit edges: Every vertex $v \in V$ such that $r(v) \in A$ (the set of all signers) is adjacent to a unique vertex $v_\mathrm{T} \in V$, $r(v_\mathrm{T}) = \mathrm{T}$. The communication with T is represented by δ. The set of vertices with outgoing signature messages is denoted by *SigSet*.

Definition 9. *Let $\mathcal{P} = (V, E, r, \mu, \delta)$ be a protocol specification, $A \subset R$ be a finite set of signers, $\mathfrak{C}$ be a contract, and SigSet $\subseteq V$. $\mathcal{P}$ is called a* DAG MPCS protocol specification for $\mathfrak{C}$, *if* [2]

1. $\exists!\ v_\mathrm{T} \in V : r(v_\mathrm{T}) = \mathrm{T} \wedge \forall v \in V \setminus \{v_\mathrm{T}\} : (v, v_\mathrm{T}) \in E$,
2. $\forall v, w \in V : v \prec w \Rightarrow (v, w) \in E \vee \exists u \in V : v \prec u \prec w \wedge r(u) \in \{r(v), r(w)\}$,
3. $\forall (v, w) \in E : \mu(v, w) =$

$$\begin{cases} \varepsilon, & \text{if } r(v) = r(w), \\ \mathit{exit}, & \text{if } w = v_\mathrm{T}, \\ \mathcal{S}_{r(v)}(\mathfrak{C}), & \text{if } v \in \mathit{SigSet} \wedge r(v) \neq r(w) \neq \mathrm{T}, \\ \wp_{r(v)}(\mathfrak{C}, v, r(w), \mathrm{T}), & \text{else.} \end{cases}$$

4. $\delta : M^* \rightarrow \left\{\text{“abort”}, (\mathcal{S}_P(\mathfrak{C}))_{P \in A}\right\}$, *where $(\mathcal{S}_P(\mathfrak{C}))_{P \in A}$ denotes a list of signatures on $\mathfrak{C}$, one by each signer.*

We write SigSet($\mathcal{P}$) for the largest subset of SigSet which satisfies

$$v \in \mathit{SigSet}(\mathcal{P}) \Rightarrow \exists w \in V : (v, w) \in E, \mu(v, w) \in M.$$

The set SigSet($\mathcal{P}$) is called the signing set.

We represent DAG MPCS protocols as *skeletal* graphs as shown in Figure 2a. The full graph, shown in Figure 2b, is obtained from the skeletal graph by adding all edges required by condition 2 of Definition 9 and extending μ according to condition 3. The ε edges are dashed in the graphs. The shaded vertices in the graphs indicate the vertices that are in the signing set. We define the *knowledge* $K(v)$ of a vertex v to be the set of message edges causally preceding v, and incoming to a vertex of the same role. The knowledge of a vertex represents the state right after its receive event.

$$K(v) = \{(w, v') \in E \mid \mu(w, v') \in M, v' \preceq v, r(v') = r(v)\}$$

We define the *pre-knowledge* $K_\prec(v)$ of a vertex v to be $K_\prec(v) = \{(w, v') \in K(v) \mid v' \prec v\}$. The pre-knowledge represents the state just *before* the vertex' receive event has taken place. We extend both definitions to sets $S \subseteq V$:

$$K(S) = \bigcup_{v \in S} K(v) \quad \text{and} \quad K_\prec(S) = \bigcup_{v \in S} K_\prec(v).$$

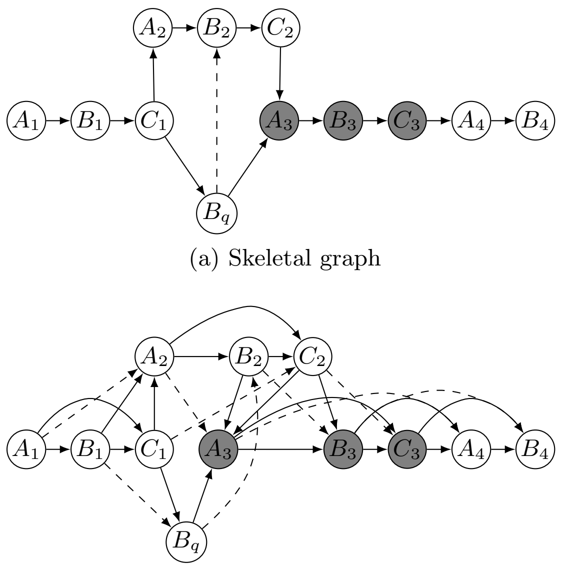
We define the *initial set* of $\mathcal{P}$, denoted *InitSet*($\mathcal{P}$) to be the set of vertices of the protocol specification for which the pre-knowledge of the same role does not contain an incoming edge by every other role. Formally,

$$v \in \mathit{InitSet}(\mathcal{P}) \iff \{r(w) \in A \mid (w, v') \in K_\prec(v)\} \cup \{r(v)\} \neq A$$

The *end set* of $\mathcal{P}$, denoted *EndSet*($\mathcal{P}$), is the set of vertices of the protocol specification at which the corresponding signer possesses all signatures.

$$v \in \mathit{EndSet}(\mathcal{P}) \iff \{r(w) \in A \mid (w, v') \in K(v), w \in \mathit{SigSet}(\mathcal{P})\} \cup \{r(v)\} = A$$

[2] We write $\exists!$ for unique existential quantification.



(a) Skeletal graph

(b) Full graph

Fig. 2. Skeletal and full representation of a DAG MPCS protocol

5.2 Resolve Protocol

Let $\mathcal{P} = (V, E, r, \mu, \delta)$ be a DAG MPCS protocol specification. The resolve protocol is a two-message protocol between a signer and the TTP T, initiated by the signer. The communication channels for this protocol are assumed to be resilient, confidential, and authentic. T is assumed to respond immediately to the signer. This is modeled in $\mathcal{P}$ via an exit edge from a vertex $v \in V \setminus \{v_\mathrm{T}\}$ to the unique vertex $v_\mathrm{T} \in V$. T's response is given by the δ function, $\delta : M^* \to \{\text{“abort”}, (\mathcal{S}_P(\mathfrak{C}))_{P \in A}\}$. If $m_1, \ldots, m_n$ is the sequence of messages sent by the signers to T, then $\delta(m_1, \ldots, m_n)$ is T's response for the last signer in the sequence. The function will be stated formally in Definition 10.

We denote the resolve protocol in the following by $\mathit{Res}(\mathfrak{C}, v)$. The signer initiating $\mathit{Res}(\mathfrak{C}, v)$ is $r(v)$. He sends the list of messages assigned to the vertices in his pre-knowledge $K_\prec(v)$, prepended by $\wp_{r(v)}(\mathfrak{C}, v, r(v), \mathrm{T})$, to T. This demonstrates that $r(v)$ has executed all receive events causally preceding v. We denote $r(v)$'s message for T by p_v:

$$p_v = \big(\wp_{r(v)}(\mathfrak{C}, v, r(v), \mathrm{T}), (\mu(w, v'))_{(w,v') \in K_\prec(v)}\big) \tag{1}$$

The promise $\wp_{r(v)}(\mathfrak{C}, v, r(v), \mathrm{T})$, which is the first element of p_v, is used by T to extract the contract $\mathfrak{C}$, to learn at which step in the protocol $r(v)$ claims to be, and to create a signature on behalf of $r(v)$ when necessary. All messages received from the signers are stored. T performs a deterministic decision procedure, shown in Algorithm 1, on the received message and existing stored messages and *immediately* sends back “abort” or the list of signatures $(\mathcal{S}_P(\mathfrak{C}))_{P \in A}$.

Our decision procedure is based on [10, 17]. The input to the algorithm consists of a message m received by the T from a signer and state information which is maintained by T. T extracts the contract and the DAG MPCS protocol specification from m. For each contract $\mathfrak{C}$, T maintains the following state information. A list $\mathit{Evidence}_{\mathfrak{C}}$ of all messages received from signers, a set $I_{\mathfrak{C}}$ of vertices the signers contacted T from, a set $\mathit{Dishonest}_{\mathfrak{C}}$ of signers considered to be dishonest, and the last decision made $\mathit{decision}_{\mathfrak{C}}$. If T has not been contacted by any signer regarding contract $\mathfrak{C}$, then $\mathit{decision}_{\mathfrak{C}} =$ "abort". Else, $\mathit{decision}_{\mathfrak{C}}$ is equal to "abort" or the list $(\mathcal{S}_Q(\mathfrak{C})_{Q \in A})$ of signatures on $\mathfrak{C}$, one by each signer.

T verifies that the request is legitimate in that the received message m is valid and the requesting signer P is not already considered to be dishonest. If these preliminary checks pass, the message is appended to $\mathit{Evidence}_{\mathfrak{C}}$. This is described in Algorithm 1 in lines 1 through 9. The main part of the algorithm, starting at line 10, concerns the detection of signers who have continued the main protocol execution after executing the resolve protocol. If P has not received a promise from every other signer in the protocol (i.e. the if clause in line 10 is not satisfied), then T sends back the last decision made (line 17). This decision is an "abort" token unless T has been contacted before and decided to send back a signed contract. If P has received a promise from every other signer, T computes the set of dishonest signers (lines 11 through 13) by adding to it every signer which has carried out the resolve protocol, but can be seen to have continued the protocol execution (line 12) based on the evidence the TTP has collected. If P is the only honest signer that has contacted T until this point in time, the decision is made to henceforth return a signed contract.

Definition 10. *Let $\mathcal{P} = (V, E, r, \mu, \delta)$ be a DAG MPCS protocol specification and δ_0 the TTP decision procedure from Algorithm 1. Then $\delta : M^* \to M$ is defined for $m_1, \ldots, m_n \in M$ by*

$$\delta(m_1, \ldots, m_n) = \pi_1(\delta_1(m_1, \ldots, m_n)),$$

where π_1 is the projection to the first coordinate and δ_1 is defined inductively by

$$\delta_1() = (\text{"abort"}, \text{"abort"}, \emptyset, \emptyset, \emptyset)$$
$$\delta_1(m_1, \ldots, m_n) = \delta_0(m_n, \delta_1(m_1, \ldots, m_{n-1})).$$

Thus the δ function represents the response of the TTP in the $\mathit{Res}(\mathfrak{C}, v)$ protocol for all executions of $\mathcal{P}$.

6 Fairness

We say that a DAG MPCS protocol execution is fair for signer P if one of the following three conditions is true: (i) No signer has received a signature of P; (ii) P has received signatures of all other signers; (iii) P has not received an "abort" token from the TTP.

The key problem in formalizing these conditions is to capture under which circumstances the TTP responds with an "abort" token to a request by a signer.

Algorithm 1. TTP decision procedure δ_0

```
input  : $m, r, decision_{\mathfrak{C}}, Evidence_{\mathfrak{C}}, I_{\mathfrak{C}}, Dishonest_{\mathfrak{C}}$
output: $r, decision_{\mathfrak{C}}, Evidence_{\mathfrak{C}}, I_{\mathfrak{C}}, Dishonest_{\mathfrak{C}}$
1  if $m \neq (\wp_P(\mathfrak{C}, v, P, \mathrm{T}), list)$ then
2      $r =$ "abort";
3      return output;
4  if $P \in Dishonest_{\mathfrak{C}} \vee \forall w \in V : m \neq p_w \vee \exists w' \in I_{\mathfrak{C}} : P = r(w')$ then
5      $Dishonest_{\mathfrak{C}} := Dishonest_{\mathfrak{C}} \cup \{P\}$;
6      $r =$ "abort";
7      return output;
8  $I_{\mathfrak{C}} := I_{\mathfrak{C}} \cup \{v\}$;
9  $Evidence_{\mathfrak{C}} := (Evidence_{\mathfrak{C}}, m)$;
10 if $v \notin InitSet(\mathcal{P})$ then
11     for $w \in I_{\mathfrak{C}}$ do
12         if $w \prec (w', x) \in K_{\prec}(I_{\mathfrak{C}}) \wedge r(w') = r(w)$ then
13             $Dishonest_{\mathfrak{C}} := Dishonest_{\mathfrak{C}} \cup \{r(w)\}$;
14     if $\forall w \in I_{\mathfrak{C}} : r(w) \notin Dishonest_{\mathfrak{C}} \implies r(w) = P$ then
15         $decision_{\mathfrak{C}} := (\mathcal{S}_Q(\mathfrak{C}))_{Q \in A}$;
16 $r = decision_{\mathfrak{C}}$;
17 return output;
```

The TTP's response is dependent on the decision procedure which in turn depends on the order in which the TTP is contacted by the signers. Since the decision procedure is deterministic, it follows that the δ function can be determined for every execution $\rho = [s_0, \alpha_1, s_1, \ldots, s_n]$ by considering the pre-knowledge of the vertices from which the *exit* transitions are taken. Abusing notation, we will write $\delta(\rho)$ instead of $\delta(m_1, \ldots, m_k)$ where m_i are the messages sent to the TTP at the i-th *exit* transition in the execution.

Definition 11. *Let* T *be the TTP. An execution* $\rho = [s_0, \alpha_1, \ldots, s_n]$ *of* $\mathcal{P}$ *is* fair *for signer* P *if one of the following conditions is satisfied:*

1. *P has not sent a signature and no signer has received signatures from* T.
$$\delta(\rho) = \text{"abort"} \wedge \forall (v, w) \in s_n : r(v) = P, r(w) \neq P \Longrightarrow v \notin SigSet(\mathcal{P})$$
2. *P has received signatures from all other signers.*
$$\exists v \in s \cap EndSet(\mathcal{P}) : r(v) = P$$
3. *P has not received an "abort" token from* T.
$$\exists (v, w) \in s : r(v) = P \wedge r(w) = \mathrm{T} \Rightarrow \delta([s_0, \ldots, s_k, exit, s_k \cup \{(v, w)\}]) \neq \text{"abort"}$$

If none of these conditions are satisfied, the execution is unfair for P.

Definition 12. *An MPCS protocol specification $\mathcal{P}$ is said to be fair, if every execution ρ of $\mathcal{P}$ is fair for all signers that are honest in ρ.*

6.1 Sufficient and Necessary Conditions

By the TTP decision procedure, T returns an "abort" token if contacted from a vertex $v \in \mathit{InitSet}(\mathcal{P})$. Thus a necessary condition for fairness is that an honest signer executes all steps of the initial set causally before all steps of the signing set that are not in the end set:

$$\forall v \in \mathit{InitSet}(\mathcal{P}), w \in \mathit{SigSet}(\mathcal{P}) \setminus \mathit{EndSet}(\mathcal{P}) : r(v) = r(w) \implies v \prec w \quad (2)$$

If P contacts T from a vertex $v \notin \mathit{InitSet}(\mathcal{P})$, then T responds with an "abort" token if it has already issued an "abort" token to another signer who is not in the set $\mathit{Dishonest}_{\mathfrak{C}}$. This condition can be exploited by a group of dishonest signers in an *abort chaining attack* [16]. The following definition states the requirements for a successful abort chaining attack. For ease of reading, we define the predicate $\mathrm{hon}(v, I)$. The predicate is true if there is no evidence (pre-knowledge) at the vertices in I that the signer $r(v)$ has sent a message at or causally after v:

$$\mathrm{hon}(v, I) \equiv \neg\exists (x, y) \in K_{\prec}(I) : v \prec (x, y) \wedge r(v) = r(x)$$

This is precisely the criterion used by T to verify honesty in Algorithm 1, line 12.

Definition 13. *Let $\mathfrak{C}$ be a contract and $l \leq |A|$. A sequence $(v_1, \dots, v_l \mid s)$ over V is called an abort-chaining sequence (AC sequence) for $\mathcal{P}$ if the following conditions hold:*

1. *Signer $r(v_1)$ receives an abort token: $v_1 \in \mathit{InitSet}(\mathcal{P})$*
2. *No signer contacts* T *more than once: $\forall_{i \neq j}\ r(v_i) \neq r(v_j)$*
3. *The present and previous signer to contact* T *are considered honest by* T*:*

$$\forall i \leq l : \mathrm{hon}(v_i, \{v_1, \dots, v_i\}) \wedge \mathrm{hon}(v_{i-1}, \{v_1, \dots, v_i\})$$

4. *The last signer to contact* T *has not previously received all signatures:*

$$\forall v \prec v_l : r(v) = r(v_l) \implies v \notin \mathit{EndSet}(\mathcal{P})$$

5. *The last signer to contact* T *has sent a signature before contacting* T *or in a parallel thread:*

$$s \in \mathit{SigSet}(\mathcal{P}) \setminus \mathit{EndSet}(\mathcal{P}) : r(s) = r(v_l) \wedge v_l \not\preceq s$$

The AC sequence represents the order in which signers execute the resolve protocol with T. A vertex v_i in the sequence implies an exit transition via the edge (v_i, v_{T}) in the protocol execution. An abort chaining attack must start at a step in which T has no choice but to respond with an abort token due to lack of information. Condition 1 covers this. Each signer may run the resolve protocol at most once. This is covered by Condition 2. To ensure that T continues to issue "abort" tokens, Condition 3 requires that there must always be a signer which according to T's evidence has not continued protocol execution after contacting T. To complete an abort chaining attack, there needs to be a signer which has issued a signature (Condition 5), but has not received a signature (Conditions 4 and 5) and will not receive a signed contract from T because there is an honest signer (by Condition 3) which has received an "abort" token.

It is not surprising that a protocol with an AC sequence is unfair. However, the converse is true, too. The proof of this and the following theorems is given in [14].

Theorem 1. *Let $\mathcal{P}$ be a DAG MPCS protocol. Then $\mathcal{P}$ is fair if and only if it has no AC sequences.*

6.2 Fairness Criteria

Theorem 1 reduces the verification of fairness from analyzing all executions to verifying that there is no AC-sequence (Definition 13). This, however, is still difficult to verify in general. The following two results are tools to quickly assess fairness of DAG MPCS protocols. The first is an unfairness criterion and the second is a fairness criterion for a large class of DAG MPCS protocols.

The following theorem states that in a fair DAG MPCS protocol, the union of paths from the initial set to every vertex $v \in \mathit{SigSet}(\mathcal{P})$ must contain all permutations of all signers (other than $r(v)$) as subsequences. In the class of linear MPCS protocols, considered in [10], this criterion was both necessary and sufficient. We show in Example 2 below that this criterion is not sufficient for fairness of DAG MPCS protocols.

For $I \subseteq V$, $v \in V$, let $\mathrm{path}(I, v)$ be the set of all directed paths from a vertex in I to v and let $\mathrm{seq}(I, v) = \{r(p) \in A^* \mid p \in \mathrm{path}(I, v)\}$ be the sequences of signers corresponding to the paths from I to v, where $r(p) = (r(v_1), \ldots, r(v_k))$.

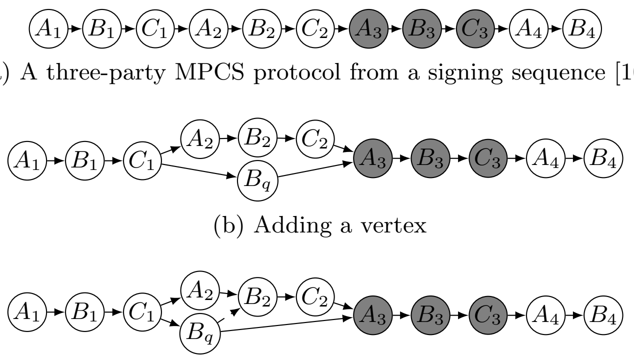
Theorem 2. *Let $k = |A|$. Let $\mathcal{P}$ be an optimistic fair DAG MPCS protocol,*

$$I = \{v \in \mathit{InitSet}(\mathcal{P}) \mid (v \prec w \wedge r(v) = r(w)) \Rightarrow w \notin \mathit{InitSet}(\mathcal{P})\} .$$

If $v \in \mathit{SigSet}(\mathcal{P})$, then for every permutation $(P_1, \ldots, P_{k-1})$ of signers in $A \setminus \{r(v)\}$, there exists a sequence in $\mathrm{seq}(I, v)$ *which contains $(P_1, \ldots, P_{k-1})$ as a (not necessarily consecutive) subsequence.*

The converse of the theorem is not true as the following example shows. In particular, this example demonstrates that the addition of a vertex to a fair DAG MPCS protocol does not necessarily preserve fairness.

Example 2. The protocol in Figure 3a is fair by the results of [10]. By Theorem 2, for every vertex $v \in \mathit{SigSet}(\mathcal{P})$ every permutation of signers in $A \setminus \{P\}$ occurs as a subsequence of a path in $\mathrm{seq}(I, v)$. The protocol in Figure 3b is obtained by adding the vertex B_q as a parallel thread of signer B. Thus the permutation property on the set of paths is preserved, yet the protocol is not fair: An AC sequence is $(B_q, C_3, A_4 | A_3)$. The vertex B_q is in $\mathit{InitSet}(\mathcal{P})$, the evidence presented to the TTP at C_3 includes the vertices causally preceding C_2, thus B is considered to be honest. The evidence presented by signer A at A_4 are the vertices causally preceding A_3 proving that B is dishonest, but C is honest. Thus A has sent a signature at A_3 but will not receive signatures from B and C.


A_1 B_1 C_1 A_2 B_2 C_2 A_3 B_3 C_3 A_4 B_4
A_1 B_1 C_1 A_2 B_2 C_2 B_q A_3 B_3 C_3 A_4 B_4
A_1 B_1 C_1 A_2 B_q B_2 C_2 A_3 B_3 C_3 A_4 B_4

(a) A three-party MPCS protocol from a signing sequence [10].

(b) Adding a vertex

(c) Adding an ε edge.

Fig. 3. Skeletal graphs of fair protocols (a, c) and an unfair protocol (b)

If a protocol has no in-role parallelism, then the converse of Theorem 2 is true. Thus we have a simple criterion for the fairness of such protocols.

Theorem 3. *Let $\mathcal{P}$ be an optimistic DAG MPCS protocol without in-role parallelism. Let*

$$I = \{v \in \mathit{InitSet}(\mathcal{P}) \mid (v \prec w \wedge r(v) = r(w)) \Rightarrow w \notin \mathit{InitSet}(\mathcal{P})\} .$$

If all paths from I to $v \in \mathit{SigSet}(\mathcal{P})$ contain all permutations of $A \setminus \{r(v)\}$ then $\mathcal{P}$ is fair for $r(v)$.

Example 3. By adding a causal edge between vertex B_q and vertex B_2 of the protocol in Figure 3b, as shown in Figure 3c, we obtain again a fair protocol.

7 Protocols

In this section we illustrate the theory and results obtained in the preceding sections by proving optimality results and constructing a variety of protocols.

7.1 Minimal Complexity

We prove lower bounds for the two complexity measures defined in our model, viz. parallel and message complexity.

Theorem 4. *The minimal parallel complexity for an optimistic fair DAG MPCS protocol is $n + 1$, where n is the number of signers in the protocol.*

Proof. By Theorem 2, every permutation of signers in the protocol must occur as a subsequence in the set of paths from a causally last vertex in the initial set to a vertex in the signing set. Since a last vertex v in the initial set must have a non-empty knowledge $K(v)$, there must be a message edge causally preceding v.

There are at least $n-1$ edges in the path between the vertices associated with the n signers in a permutation and there is at least one message edge outgoing from a vertex in the signing set. Thus a minimal length path for such a protocol must contain $n+1$ edges.

The minimal parallel complexity is attained by the broadcast protocols of Baum-Waidner and Waidner [2].

Theorem 5. *The minimal message complexity for an optimistic fair DAG MPCS protocol is $\lambda(n) + 2n - 3$, where n is the number of signers in the protocol and $\lambda(n)$ is the length of the shortest sequence which contains all permutations of elements of an n-element set as subsequences.*

The minimal message complexities for $2 < n < 8$ are $n^2 + 1$. The minimal message complexities for $n \geq 10$ are smaller or equal to n^2.

Note that while broadcasting protocols have a linear parallel complexity, they have a cubic message complexity, since in each of the $n+1$ rounds each of the n signers sends a message to every other signer. Linear protocols on the other hand have quadratic minimal message and parallel complexities. In the following we demonstrate that there are DAG protocols which attain a linear parallel complexity while maintaining a quadratic message complexity.

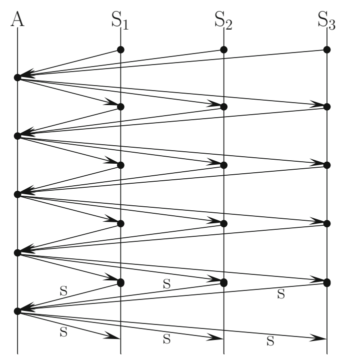
7.2 Protocol Constructions

Single Contractor, Multiple Subcontractors. A motivation for fair MPCS protocols given in [10] is a scenario where a single entity, here referred to as a contractor, would like to sign k contracts with k independent companies, in the following referred to as subcontractors. The contractor has an interest in either having all contracts signed or to not be bound by any of the contracts. The subcontractors have no contractual obligations towards each other. It would therefore be advantageous if there is no need for the subcontractors to directly communicate with each other.

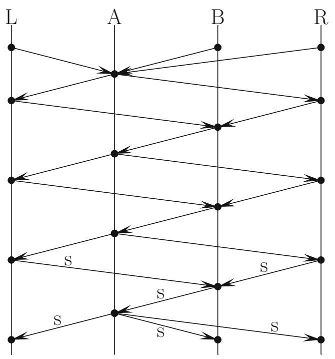
The solutions proposed in [10] are linear protocols. Their message and parallel complexities are thus quadratic. Linear protocols can satisfy the requirement that subcontractors do not directly communicate with each other only by greatly increasing the message and parallel complexities.

The protocol we propose here is a DAG, its message complexity is $2(n+1)(n-1)$ and its parallel complexity is $2n+2$ for n signers. It therefore combines the low parallel complexity typically attained by broadcasting protocols with the low message complexity of linear protocols. Additionally, the protocol proposed does not require any direct communication between subcontractors.

Figure 4a shows a single contractor with three subcontractors. The protocol can be subdivided into five rounds, one round consisting of the subcontractors sending a message to the contractor followed by the contractor sending a message



(a) A single contractor and three subcontractors.



(b) Two joint subcontractors.

Fig. 4. Two examples of fair DAG MPCS protocols

to the subcontractors. In the first four rounds promises are sent, in the final round signatures are sent. The protocol can be easily generalized to more than three subcontractors. For every subcontractor added, one extra round of promises needs to be included in the protocol specification.

The protocol is fair by Theorem 3. The MSC shown in Figure 4a resembles the skeletal graph from which it was built. The message contents can be derived by computing the full graph according to Condition 2 of Definition 9. The result is as follows. In each round of the protocol, each of the subcontractors sends to the contractor a promise for the contractor and for each of the other subcontractors. The contractor then sends to each of the subcontractors all of the promises received and his own promise. The final round is performed in the same manner, except that promises are replaced by signatures.

Two Contractors with Joint Subcontractors. Figure 4b shows a protocol where two contractors want to sign a contract involving two subcontractors. The subcontractors are independent of each other.

After the initial step, where all signers send a promise to the first contractor A, there are three protocol rounds, one round consisting of the contractor A sending promises to the two subcontractors L and R in parallel which in turn send promises to the second contractor B. A new round is started with the second contractor sending the promises received with his own promise to contractor A.

This protocol, too, can be generalized to several independent subcontractors. For every subcontractor added, one extra protocol round needs to be included in the protocol specification and each protocol step of the subcontractors executed analogously.

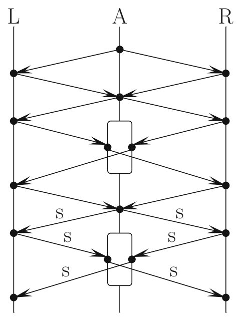


Fig. 5. In-role parallelism

Parallelism within a Role. Figure 5 shows an example of a subcontracting protocol with in-role parallelism for the contractor role. The contractor initiates the protocol. In the indicated parallel phase, the contractor may immediately forward a promise by one of the subcontractors along with his own promise to the other subcontractor without waiting for the latter subcontractor's promise. The same is true in the signing phase. The fairness property for this protocol has been verified with a tool which tested fairness for each signer in all possible executions.

8 Conclusion

We have identified fair subcontracting as a challenging new problem in the area of multi-party contract signing. We have made first steps towards solving this problem by introducing DAG MPCS protocols and extending existing fairness results from linear protocols to DAG protocols. For three typical subcontracting configurations we propose novel DAG MPCS protocols that perform well in terms of message complexity and parallel complexity. Fairness of our protocol schemes follows directly from our theoretical results and we have verified it for concrete protocols with our automatic tool.

There are a number of open research questions related to fair subcontracting that we haven't addressed. We mention two. The first concerns the implementation of multi-contracts. In our current approach we consider a single abstract contract shared by all parties. However, in practice such a contract may consist of a number of subcontracts that are accessible to the relevant signers only. How to cryptographically construct such contracts and what information these contracts should share is not evident. Second, a step needs to be made towards putting our results into practice. Given the application domains identified in this paper, we must identify the relevant signing scenarios and topical boundary conditions in order to develop dedicated protocols for each application area.

Acknowledgement. We thank Barbara Kordy for her many helpful comments on this paper.

References

1. Asokan, N.: Fairness in electronic commerce. PhD thesis, Univ. of Waterloo (1998)
2. Baum-Waidner, B., Waidner, M.: Optimistic asynchronous multi-party contract signing. Research Report RZ 3078 (#93124), IBM Zurich Research Laboratory, Zurich, Switzerland (November 1998)

3. Baum-Waidner, B., Waidner, M.: Round-optimal and abuse free optimistic multi-party contract signing. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 524–535. Springer, Heidelberg (2000)
4. Chadha, R., Kremer, S., Scedrov, A.: Formal analysis of multi-party contract signing. In: CSFW 2004, p. 266. IEEE, Washington, DC (2004)
5. Even, S., Yacobi, Y.: Relations among public key signature systems. Technical Report 175, Computer Science Dept. Technion, Haifa, Isreal (March 1980)
6. Garay, J.A., Jakobsson, M., MacKenzie, P.D.: Abuse-free optimistic contract signing. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 449–466. Springer, Heidelberg (1999)
7. Garay, J.A., MacKenzie, P.D.: Abuse-free multi-party contract signing. In: Jayanti, P. (ed.) DISC 1999. LNCS, vol. 1693, pp. 151–166. Springer, Heidelberg (1999)
8. Karaenke, P., Kirn, S.: Towards model checking & simulation of a multi-tier negotiation protocol for service chains. In: Int. Found. for Autonomous Agents and Multiagent Systems AAMAS 2010, pp. 1559–1560 (2010)
9. Katok, E., Pavlov, V.: Fairness in supply chain contracts: a laboratory study. J. of Operations Management 31, 129–137 (2013)
10. Kordy, B., Radomirović, S.: Constructing optimistic multi-party contract signing protocols. In: CSF 2012, pp. 215–229. IEEE Computer Society (2012)
11. Kraus, S.: Automated negotiation and decision making in multi-agent environments. In: ACM Multi-agent Systems and Applications, pp. 150–172 (2001)
12. Krishnan, H., Winter, R.: The economic foundations of supply chain contracting. Foundations and Trends in Technology, Information and Operations Management 5(3-4), 147–309 (2012)
13. Lu, K., Yahyapour, R., Yaqub, E., Kotsokalis, C.: Structural optimisation of reduced ordered binary decision diagrams for SLA negotiation in IaaS of cloud computing. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC 2012. LNCS, vol. 7636, pp. 268–282. Springer, Heidelberg (2012)
14. Mauw, S., Radomirović, S.: Generalizing Multi-party Contract Signing. CoRR, abs/1501.03868 (Extended version.) (2015), `http://arxiv.org/abs/1501.03868`
15. Mauw, S., Radomirović, S., Dashti, M.T.: Minimal message complexity of asynchronous multi-party contract signing. In: CSF 2009, pp. 13–25. IEEE (2009)
16. Mukhamedov, A., Ryan, M.D.: Improved multi-party contract signing. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886, pp. 179–191. Springer, Heidelberg (2007)
17. Mukhamedov, A., Ryan, M.D.: Fair multi-party contract signing using private contract signatures. Inf. Comput. 206(2-4), 272–290 (2008)
18. Schunter, M.: Optimistic Fair Exchange. Phd thesis, Universität des Saarlandes (2000)
19. Seifert, R., Zequiera, R., Liao, S.: A three-echelon supply chain with price-only contracts and sub-supply chain coordination. Int. J. of Production Economics 138, 345–353 (2012)
20. Yaqub, E., Wieder, P., Kotsokalis, C., Mazza, V., Pasquale, L., Rueda, J., Gómez, S.G., Chimeno, A.: A generic platform for conducting SLA negotiations. In: Service Level Agreements for Cloud Computing, pp. 187–206. Springer (2011)
21. Zhang, Y., Zhang, C., Pang, J., Mauw, S.: Game-based verification of contract signing protocols with minimal messages. Innovations in Systems and Software Engineering 8(2), 111–124 (2012)