An Intruder Model for Verifying Liveness in Security Protocols

Jan Cederquist
Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
Jan.Cederquist@cs.utwente.nl

Mohammad Torabi Dashti
Centrum voor Wiskunde en Informatica
P.O. Box 94079, NL-1090 GB Amsterdam
The Netherlands
Mohammad.Dashti@cwi.nl

ABSTRACT

We present a process algebraic intruder model for verifying a class of liveness properties of security protocols. For this class, the proposed intruder model is proved to be equivalent to a Dolev-Yao intruder that does not delay indefinitely the delivery of messages. In order to prove the equivalence, we formalize the resilient communication channels assumption. As an application of the proposed intruder model, formal verification of fair exchange protocols is discussed.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Verification*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal Methods, Model Checking*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General Terms

Security, Theory, Verification

Keywords

Intruder Models, Security Protocols, Liveness Properties, Dolev-Yao Intruder

1. INTRODUCTION

Liveness aspects of security protocols, stipulating that some desired situation eventually will occur, have only recently found a role on the scene with the requirements emerging from electronic commerce applications. Fair payment, certified email, non-repudiation and electronic contract signing protocols are typical examples of relevant applications (see e.g. [4, 27, 37]). These protocols all aim at a liveness requirement, namely fair exchange of valuable electronic items (money, signature, evidence etc).

The liveness requirements of fair exchange protocols clearly cannot be achieved unless messages are eventually delivered over (at least some) communication channels. However, a communication channel that eventually delivers transmitted messages (called a *resilient* channel [4]) deviates from the de-facto standard model used in analyzing security protocols. Following the seminal work of Dolev and Yao [16], the communication media are assumed to be under absolute control of the intruder. This intruder can in particular destroy transmitted messages.

We aim at automatic verification of liveness properties of security protocols. To use the standard Dolev-Yao (DY) intruder thus requires a fairness constraint that prevents the intruder from disrupting communications over resilient channels. Due to the complexity of such a fairness constraint, automatic verification of liveness properties in the DY model seems to be very inefficient (as will be motivated in this paper).

Checking liveness aspects of fair exchange protocols has often been left out from formal analysis (see [1, 9, 26, 34] for some notable studies). There are however at least two reasons why it is crucial to check some liveness properties. Let us refer to them as the "theoretical reason" and the "pragmatic reason". The theoretical reason is that fair exchange inherently contains a liveness property. This has also been confirmed in practice as the non-termination flaws reported in [7, 23, 35], respectively in protocols of [6, 37, 36], would not have been found if liveness of participants had been taken for granted. The pragmatic reason is that in models of protocols certain behaviors might be missing or differently represented in their realizations. In particular, security protocols do in practice only have finite runs (e.g. there are always time-out operations) and safety properties (stating that undesirable states are not reachable) may seem sufficient. However, in modeling security protocols, timing aspects are usually abstracted away (see [30] for a survey). For example, if a loop is mistakenly specified in Apache's rewrite module [19], theoretically there can be a *livelock* in the server model that should be detected, even if the bogus server is not in practice endlessly trapped. As another example, a typical requirement for a fair exchange protocol between Alice and Bob states that, if Alice receives a desired item from Bob, then Bob will eventually receive his desired

FMSE'06, November 3, 2006, Alexandria, Virginia, USA.

item from Alice as well [4]. This is a liveness property that does not hold in the protocol model if there is either a *deadlock* situation, such that Bob cannot progress after Alice has got her item, or a *livelock* situation, such that Bob runs into an endless execution (possibly maliciously devised to prevent him to ever reach his goal). The latter situation sets up a threat alarm, although the realization does not necessarily contain a livelock, i.e. Bob at some point has to give up the exchange in an unfair state.

Contributions In this paper we investigate the suitability of the DY intruder model for automatic verification of liveness properties under the resilient communication channels (RCC) assumption. First, we present a formalization of RCC. This formalization cannot be expressed in linear temporal logic (LTL [33]), which indicates that using RCC as a fairness constraint[1] in verifying liveness properties would not be efficient. Therefore, we present a process algebraic intruder model, restricted by a much simpler fairness constraint, that is proved to be *equivalent* to a DY intruder that does not indefinitely delay the delivery of messages: For an arbitrary protocol and a liveness property ϕ of the general form "inevitable reachability of a particular action in all *fair* traces", we prove that if the DY intruder finds an attack (counterexample) for the property, without violating the RCC assumption, then the proposed intruder model can find a corresponding attack, and vice versa. For safety properties, which do not require the RCC assumption, our intruder model is equivalent to the DY intruder, modulo action renaming. Finally, we illustrate how fair exchange properties can be expressed in terms of ϕ and a few safety properties, which can all be checked in the proposed intruder model.

Structure of the paper Section 2 contains definitions and notations that are used later in the paper. In section 3 we present a formalization of the RCC assumption. In section 4 we propose an intruder model that is proved to be equivalent to a DY intruder that respects the RCC assumption. The equivalence proof is sketched in appendix A. As an application of the proposed intruder, formal verification of fair exchange protocols is discussed in section 5. Section 6 discusses some related work and section 7 concludes.

2. PRELIMINARIES

In this section we briefly recall general concepts and notations used in the paper.

2.1 The μCRL Process Algebra

We use the process algebraic language μCRL [22], which is an extension of ACP [10] with abstract data types, for specifying protocols and intruders. Our results do not depend however on this choice, as μCRL is similar to other general purpose process algebras, such as CSP [25], and is used here merely to provide a precise description of the presented specifications. A μCRL specification describes a labeled transition system (see section 2.2), in which states represent process terms and edges are labeled with actions. What follows only gives a brief description of μCRL, while its complete syntax and semantics is given in [22].

[1]Two notions of fairness are used in this paper: fairness in exchange (section 5) and fairness constraints of a labeled transition system. The latter is used to describe "fair" execution traces.

In μCRL a specification consists of data type declarations and process behavior definitions, where processes and actions can be parameterized with data. We assume the presence of a data sort *Bool* of booleans with constants T and F, and the usual connectives $\wedge$, $\vee$ and $\neg$.

The specification of a process is constructed from actions belonging to a finite set *Act* which contains a deadlock action δ (denoting that from then on, no action can be performed) and a silent internal action τ, recursion variables and process algebraic operators. Processes are represented by process terms that describe the order in which actions may happen. A process term consists of action names and recursion variables combined by process algebraic operators. The expressions $p.q$ and $p + q$ denote sequential composition and non-deterministic choice, respectively. In the intruder specifications (sections 4.2 and 4.3), the operator . has the strongest precedence and + has the weakest. The summation $\Sigma_{d:D}\ p(d)$ provides the possibly infinite choice over a data type D, and the conditional construct $p \lhd b \rhd q$, with b a data term of sort *Bool*, behaves as p if $b = \mathsf{T}$ and as q if $b = \mathsf{F}$. The construct $\Sigma_{d:D}\ p(d) \lhd f(d) \rhd \delta$, used in section 4, with f being a Boolean function of d, forces the choice of d values that satisfy f. The parallel composition $p \| q$ interleaves the actions of p and q. Moreover, actions from p and q may synchronize ($p|q$), when this is explicitly allowed by a predefined partial synchronization function $| : Act \times Act \rightarrow Act$. However, two actions can only synchronize if their data parameters are semantically the same, which means that synchronization can be used to represent data transfer between processes. Encapsulation $\partial_H(p)$, which renames all occurrences of actions from the set H in p into the deadlock action δ, can be used to force actions into communication.

2.2 Labeled Transition Systems

A labeled transition system (LTS) L is a tuple (S, s_0, Act, Tr), where S is a finite set of states, $s_0 \in S$ is the initial state, *Act* is a finite set of actions, and $Tr \subseteq S \times Act \times S$ is the transition relation. A transition $(s, a, s') \in Tr$, denoted $s \xrightarrow{a} s'$, indicates that the system can move from state s to s' by taking action a.

A sequence is a function $\alpha : I \rightarrow A$, where A is a finite set and the index set I is either $\{0, \ldots, n-1\}$ (for finite sequences of length n) or the natural numbers *Nat* (for infinite sequences). When $I = \{0, \ldots, n-1\}$, the notation I^+ is used to refer to $\{0, \ldots, n\}$. When I is *Nat*, then $I^+ = I$. The notation α_i stands for the application $\alpha\ i$.

A *trace* in L is a sequence of transitions $\alpha : I \rightarrow Tr$ for which there is a sequence of states $\sigma^\alpha : I^+ \rightarrow S$ such that $\sigma^\alpha_0 = s_0$ and $\forall i \in I,\ \exists a \in Act.\ \alpha_i = \sigma^\alpha_i \xrightarrow{a} \sigma^\alpha_{i+1}$. We loosely use α_i and its corresponding action interchangeably.

A sequence of states $\sigma : I \rightarrow S$ in L gives rise to a sequence of sets of *enabled transitions* $\hat{\sigma} : I \rightarrow 2^{Tr}$ defined as $\hat{\sigma}_i = en(\sigma_i)$, where $en(s)$ is the set of transitions that are enabled in s: $en(s) = \{(s, a, s') | s \xrightarrow{a} s'\}$.

A finite trace α is called a *deadlock* trace iff $\hat{\sigma}^\alpha_{|I|} = \emptyset$ (where $|I|$ is the cardinality of I).

In modeling, a desired property is sometimes violated only due to unrealistic behaviors in the model that do not reflect the actual system under study. In this case fairness constraints are used to rule out such behaviors, cf. [21]. A fairness constraint states whether a given trace of the model reflects an *acceptable* (or realistic) execution of the system

or not. Unacceptable traces are then excluded from subsequent (formal) analysis [21]. We are now ready to define a fairness constraint for LTSs, informally stating that no possibilities are excluded forever.

DEFINITION 1. *A trace* $\alpha : I \rightarrow Tr$ *in an LTS* $L = (S, s_0, Act, Tr)$ *is* fair[2] *iff, for each* $\theta \in Tr$*, if* $\{i \in I \mid \theta \in \hat{\sigma}_i^{\alpha}\}$ *is infinite then so is* $\{i \in I \mid \theta = \alpha_i\}$.

Note that finite traces are always fair.

2.3 Modeling a Protocol

This section describes how we model protocols and intruders.

We consider an asynchronous communication model with no global clock. A security protocol P is modeled as an asynchronous composition of a finite number of finitely-branching non-deterministic processes with names. These processes model the roles of participants in the protocol. Processes communicate by sending and receiving messages.

A message is a pair $m = \langle p, c \rangle$, where p is the identity of the intended receiver process and c is the content of the message. We let *Msg* be the set of all message contents that can be communicated (for a recursive formalization of *Msg* see e.g. [32]). Messages are exchanged via a communication network *net*, which is also seen as a process (equipped with some internal buffer), that is external to P. We overload P to denote also the set of processes which constitute it.

To send or receive a message m, an agent $p \in P$ performs the actions $\mathbf{send}(p, m)$ or $\mathbf{recv}(p, m)$, respectively. Even though an agent p sends the message m with the intention that it should be received by an agent q, it is in fact the network (the process *net*) that receives the message from p, and it is from the network that q can receive m. For each message dispatched to *net* we define a corresponding delivery as $\mathbf{send}(p, \overline{\langle q, c \rangle}) = \mathbf{recv}(q, \langle q, c \rangle)$[3]. The communications between protocol participants and *net* are assumed to be synchronized (see section 2.1), meaning that p can **send** a message m to *net* iff at the same time *net* can receive m from p, and similarly for **recv**. The communication between participants of a protocol via *net* is however asynchronous and a participant has no guarantees about the origins of the messages it receives.

Apart from **send** and **recv**, all other actions of processes in P are considered *internal* (as in [22]), i.e. do not involve any process outside P. We assume that each protocol performs two specific internal actions i and t that are used to express a certain liveness goal of the protocol. These actions are abstract and no semantics is attributed to them. All other internal actions of processes in P are considered silent (represented by τ), i.e. do not appear in the property being verified. These can denote, for instance, internal decisions or communications between protocol participants through secure channels.

To model an intruder that has complete control over the network, we assume that it plays the role of the network. The intruder may thus schedule messages and possibly insert its own messages into the network. Besides *being* the network, the intruder can also have roles in protocols (which is required, for instance, in modeling fair exchange protocols).

The merit of using synchronous communication between P and *net* (played by the intruder) is that the intruder does not try to compose messages which P does not accept, simply because synchronizations with the corresponding actions of P would fail. This allows us to define the (infinite) set *Msg* of possible messages recursively, and still be able to specify a generic intruder model whose specification does not depend on the protocol being analyzed.

In this paper, we study LTSs L that result from the interactions between an intruder and a protocol P, i.e. $L = (P \| net)$. Each state of such an L corresponds to the Cartesian product of the states of the intruder and of P.

2.4 Safety, Liveness and the DY Intruder

Properties of systems can usually be divided into two classes: *safety* properties (something bad will never happen) and *liveness* properties (something good will eventually happen). For formal definitions of safety and liveness we refer to [3]. When verifying liveness, fairness constraints usually have to be imposed on the model to rule out unrealistic executions [21]. The notation $L \models_{\mathcal{F}} \phi$ is used here to represent that the liveness property ϕ holds in an LTS L under fairness constraint $\mathcal{F}$.

Most security requirements are encoded as safety properties. *Secrecy*, for instance, can usually be expressed as "a secret item is never revealed to an outsider". These properties of security protocols are often verified using the DY intruder model [16]. The DY intruder has complete control over the network. It intercepts and remembers all messages that have been transmitted. It can decrypt, encrypt and sign messages, if it knows the corresponding key. It can decompose, compose and send new messages from its knowledge. It can also remove or delay messages in favor of others being communicated. The DY model is the most powerful intruder under the perfect cryptography assumption [13].

As an instance of liveness requirements one can think of *fair exchange* requirements, a topic to which we will return in section 5. Liveness properties do not hold in general in the presence of the DY intruder, as it can deliberately disrupt all communications. To verify liveness properties, the DY intruder model should be modified to respect the resilient communication channels (RCC) assumption[4], i.e. the behaviors of the model that violate RCC should be excluded. To verify a liveness property ϕ for a protocol P in this model, we thus need to decide $(P \| DY) \models_{RCC} \phi$.

3. FORMALIZING THE RCC ASSUMPTION

The resilient communication channels (RCC) assumption informally states that all messages transmitted over a resilient network, i.e. over an arbitrary topology of resilient channels, will eventually reach their destinations[5]. No (non-trivial) liveness property of protocols can be guaranteed in

[2]This corresponds to the strong notion of fairness in [21]: $\forall \theta \in Tr.\ F^{\infty}\ \text{enabled}(\theta) \Rightarrow F^{\infty}\ \text{executed}(\theta)$.

[3]Using q redundantly in $\mathbf{recv}(q, \langle q, c \rangle)$ ensures that the parameters of **send** and **recv** actions are semantically the same.

[4]As a side note, a DY intruder that respects *RCC* is more powerful (as in [13]) than the intruders that are often used in verifying wireless protocols (e.g see [5]), i.e. those which cannot delay or block messages, but can eavesdrop and inject self-fabricated messages.

[5]We study this asymptotic behavior since no time bound is then forced on the intruder. It can delay messages for any finite amount of time.

general without the RCC assumption (for a list of protocols assuming RCC see e.g. [27]). This is a consequence of the impossibility of distributed consensus with one faulty process [20]. Besides, a resilient network can be realized using unreliable links [8], under reasonable assumptions.

We make the following choices when formalizing RCC:

C1. A resilient network not only delivers all transmitted messages, but is also infinitely often ready to receive messages.

C2. A resilient network may "destroy" a message only when it would not be accepted by its recipient in any future point of time[6].

C3. A finite version of RCC is considered: For a given $n \in Nat$, a network that realizes RCC_n (called an RCC_n network) guarantees that, if the message is sent n' times to the network, then it will eventually deliver $min(n, n')$ instances intact to its destination (if the message has a recipient).

Clearly, RCC_n is a weaker constraint than RCC, since it accepts more behaviors. In fact, for each n, it is easy to come up with a protocol P and a liveness property ϕ such that $(P\|net) \models_{RCC} \phi$ and $(P\|net) \not\models_{RCC_n} \phi$. Nevertheless, by assuming RCC_n when checking liveness properties we are on the safe side, since if a liveness property holds in an RCC_n-network, it also holds in any RCC_m-network for $m \geq n$. Pushing this idea further, to simplify our formalization[7] we use RCC_1 from this point on.[8]

3.1 Formalizing RCC_1 Networks

Consider a protocol P running in an RCC_1-network *net*, i.e. $(P\|net)$. We define what it means for *net* (played by the intruder) to respect RCC_1, by defining a fairness constraint.

Compared to the notion of fairness in definition 1, RCC_1 is a weaker constraint because it allows executions that indefinitely ignore the delivery of a message that has not been transmitted to *net* after the last delivery of that particular message. In the definition of this fairness constraint we should thus make sure that when the intruder plays the role of the network, under RCC, it will never be forced[9] to send any intruder-fabricated message.

Below we characterize the set of actions an RCC_1-network should "treat in a fair way". This set is smaller than $en(s)$, used in definition 1. It includes all **send** actions (C1 in section 3), all **recv** actions that are not equal to any pending $\overline{\textbf{send}}$ (C2) and finally all internal actions of protocol participants.

In the following definitions we let $(P\|net)$ be represented by $L = (S, s_0, Act, Tr)$. The sequence of sets of transitions that should be treated in a fair way is defined as follows:

DEFINITION 2. *Let $\alpha : I \to Tr$ be a trace of L. The sequence $\check{\sigma}^\alpha : I^+ \to 2^{Tr}$ is defined as $\check{\sigma}^\alpha_i = \hat{\sigma}^\alpha_i \setminus \Theta^\alpha_i$, where Θ^α_i is the set of non-pending* **recv** *actions:*

$$\Theta^\alpha_i = \{\mathbf{recv}(q, \langle q, c\rangle) \in \hat{\sigma}^\alpha_i \mid \exists j < i, p \in P.\alpha_j = \mathbf{send}(p, \langle q, c\rangle) \Rightarrow \exists l \in I.j < l < i \wedge \alpha_l = \overline{\alpha}_j\}.$$

Note that $\check{\sigma}$ is defined for a particular trace. Whether a transition, enabled in a certain state, is required by RCC_1 or not, depends in general on the trace taken to reach that state. Now we can define the traces that are fair according to RCC_1:

DEFINITION 3. *An infinite trace $\alpha : I \to Tr$ belonging to L is called* RCC-fair *iff, for all $\theta \in Tr$, if $\{i \in I \mid \theta \in \check{\sigma}^\alpha_i\}$ is infinite then so is $\{i \in I \mid \theta = \alpha_i\}$. A finite trace α belonging to L is RCC-fair iff α is a deadlock trace.*

This definition characterizes a weaker notion of fairness constraint compared with definition 1, simply because $\check{\sigma}^\alpha_i \subseteq \hat{\sigma}^\alpha_i$.

3.2 Formalizing RCC_1 in Presence of the DY Intruder

In section 3.1 we implicitly assumed that a communication network would not permanently stop operating. We relax this assumption in this section since the DY intruder can take down the communication network at any moment. However, this should only be allowed in certain situations. Intuitively, the intruder may block the network only if there is nothing to receive from other processes (C1), and if all messages that it received earlier, to which there eventually will be a recipient in the protocol, have already been sent (C2). In our formalization we assume, without lack of generality, that an internal action κ occurs before the intruder permanently blocks the communication media. This symbolic action is used in the formalization of RCC_1:

DEFINITION 4. *A trace $\alpha : I \to Tr$* respects *RCC_1 in L iff α is RCC-fair and*

$$\begin{array}{l}\forall i \in I.\\ \quad \alpha_i = \kappa \Rightarrow\\ \quad\quad \neg\exists p, q \in P, c \in Msg.\\ \quad\quad\quad \mathbf{send}(p, \langle q, c\rangle) \in en^*(\sigma^\alpha_i) \wedge\\ \quad\quad \exists p, q \in P, c \in Msg, j < i.\\ \quad\quad\quad (\mathbf{recv}(q, \langle q, c\rangle) \in en^*(\sigma^\alpha_i) \wedge\\ \quad\quad\quad\quad \alpha_j = \mathbf{send}(p, \langle q, c\rangle)) \Rightarrow\\ \quad\quad\quad\quad \exists k.j < k < i \wedge \alpha_k = \overline{\alpha}_j,\end{array}$$

where $en^*(s) = \cup_{s \xrightarrow{\tau}{}^* s'} en(s')$ and $\xrightarrow{\tau}{}^*$ is the reflexive transitive closure of $\xrightarrow{\tau}$.

The set $en^*(s)$ characterizes the transitions that the processes of P can perform after an arbitrary number of internal actions.

From the security point of view it is notable that, if the DY intruder receives a message $m = \langle q, c\rangle$ from p, and manages to instead send to q the message $m' \neq m$ such that in the

[6]This may seem to introduce some spurious behaviors in the system. For instance consider two processes p and q communicating on a network, specified as: $p = i.\mathbf{send}(p, \langle q, m_1\rangle).\mathbf{send}(p, \langle q, m_2\rangle).\delta$ and $q = \mathbf{recv}(q, \langle q, m_2\rangle).\mathbf{recv}(q, \langle q, m_1\rangle).t.\delta$. One might expect that action t does not happen due to the mismatch in these specifications. In a real network, however, the messages from p can be shuffled such that q can reach t. This shuffling is also allowed in our model, considering it as a consequence of the asynchronous communication model.

[7]Modeling an RCC-network in general would require pushdown automata to track received and subsequently delivered messages, whereas RCC_1 conceptually corresponds to finite automata.

[8]Confining to RCC_1 comes to terms with practice as it is not advised to use the same message for different purposes in a protocol [2].

[9]Note that "forcing the intruder" only arises in modeling, e.g. by ignoring other possible behaviors of the model.

resulting state s, $\mathbf{recv}(q,m) \notin en^*(s)$, then RCC does not prevent the intruder from never sending m.

The following lemma states that the formalization of RCC is not expressible in LTL. It is thus questionable whether it can readily be used as fairness constraint in efficient verification.

LEMMA 1. *RCC is not expressible in LTL.*

Proof: We give two witness LTSs that are indistinguishable for LTL[10], but not for RCC:

$$L_1 = \{s_0 \xrightarrow{\tau} s_1, s_1 \xrightarrow{\kappa} s_2, s_1 \xrightarrow{\mathbf{send}} s_3\}$$
$$L_2 = \{s_0 \xrightarrow{\tau} s_1, s_0 \xrightarrow{\tau} s_2, s_1 \xrightarrow{\kappa} s_3, s_2 \xrightarrow{\mathbf{send}} s_4\}$$

■

4. AN INTRUDER FOR VERIFYING LIVENESS PROPERTIES

We consider an arbitrary protocol P with a liveness property ϕ of the general form "if a certain action i happens, another action t will eventually happen" (formalized in section 4.2). Although ϕ does not comprise all possible liveness properties, it notably covers those appearing in fair exchange protocols. Our goal is to verify whether $(P\|DY) \models_{RCC} \phi$. But, the formalization of RCC as a fairness constraint (definition 4) is not expressible in LTL (lemma 1). In order to efficiently check ϕ under the RCC assumption, we need to express the fairness constraint in the infinitary fragment of LTL [18][11]. We therefore chose to modify the DY intruder model to $I^\dagger$ such that

$$(P\|DY) \models_{RCC} \phi \;\equiv\; (P\|I^\dagger) \models_{\mathcal{F}} \phi,$$

where the fairness constraint $\mathcal{F}$ is indeed expressible in infinitary LTL.

The intruder model $I^\dagger$ is presented in section 4.3 and we study the relation between the LTSs resulting from $(P\|DY)$ and $(P\|I^\dagger)$ in section 4.4.

4.1 Regular alternation-free μ-calculus

In order to formulate properties of states in LTSs, we use the regular alternation-free μ-calculus, which is a fragment of μ-calculus that can efficiently be model-checked [29]. We now briefly present the relevant part of this logic (for a complete treatment of its syntax and semantics, we refer to [29]). This logic consists of *regular formulas* and *state formulas*.

Regular formulas, which describe sets of execution traces, are built upon *action formulas* and the standard regular expression operators. We use $\cdot$, $\vee$, $\neg$ and * for concatenation, choice, complement and transitive-reflexive closure of regular formulas, respectively.

State formulas, expressing properties of states, are built upon propositional variables, standard boolean operators, the possibility modal operator $\langle\cdots\rangle$, the necessity modal operator $[\cdots]$ and the minimal and maximal fixed point operators μ and ν. The operator $\langle\cdots\rangle$ is used in the form $\langle R\rangle\mathsf{T}$ to express the existence of an execution of the model for which the regular formula R holds, and $[\cdots]$ is used in the form $[R]$ to quantify over all executions of the model in which the regular formula R holds. A state satisfies $\mu X.f$ iff it belongs to the minimal solution of the fixed point equation $X = f(X)$, with f being a state formula and X a set of states.

The symbol T is used in both action formulas and state formulas. In action formulas it represents *any action* and in state formulas it denotes the entire state space. The wildcard action parameter _ represents any parameter of an action.

4.2 Specification

The Dolev-Yao intruder DY is seen here as a non-deterministic process which exhaustively tries all possible sequences of actions. It can, in particular, deliberately terminate (by performing κ), which corresponds to taking the network permanently down:

$$\begin{aligned} DY(X) \;=\; & \sum_{c\in Msg, p,q\in P} \mathbf{recv}_I(p,\langle q,c\rangle).DY(\{c\}\cup X) \;+ \\ & \sum_{c\in Msg, p\in P} \mathbf{send}_I(p,\langle p,c\rangle).DY(X) \\ & \quad \lhd synth(c,X) \rhd \delta \;+ \\ & \kappa.\delta \end{aligned}$$

Above, the set X of messages represents the (knowledge) state of the intruder, Msg is the (infinite) set of possible messages, and the function $synth : Msg \to 2^{Msg} \to Bool$ characterizes the ability of the DY intruder to compose new messages from its knowledge (cf. section 2.4). For an inductive formalization of *synth* we refer to [32][12]. Note that the equivalence results in section 4.4 do not depend however on *synth*.

Suppose that the intruder starts with the initial knowledge $X = K_0$ and the protocol starts in the initial state $\mathtt{P}_0$. We define $L_A \stackrel{def}{=} (S_A, s_{0A}, Act, Tr_A)$ as the LTS described by $\partial_\Delta(P(\mathtt{P}_0)\|DY(K_0))$.

We assume that the processes in P perform $\mathbf{send}_P$ and $\mathbf{recv}_P$ for sending and receiving messages, respectively. We let $\Delta = \{\mathbf{send}_I, \mathbf{recv}_I, \mathbf{send}_P, \mathbf{recv}_P\}$ and force the members of Δ to synchronize as

$$\mathbf{send}_I|\mathbf{recv}_P = \mathbf{recv}$$
$$\mathbf{recv}_I|\mathbf{send}_P = \mathbf{send}.$$

In the previous sections we assumed that processes perform **send** and **recv** for sending and receiving actions. Here we go a step further and make the intruder explicit in the model.

Finally the liveness property ϕ is expressed as

$$\phi_A \;\stackrel{def}{=}\; [\mathsf{T}^*\cdot i]\;\mu X.(\langle\mathsf{T}\rangle\mathsf{T} \;\wedge\; [\neg t]X),$$

i.e. for each i action there is a (not necessarily unique) t action coming later.[13] Then we say there is an *attack* in system A iff $L_A \not\models_{RCC} \phi_A$.

[10]To interpret LTL formulas, which are originally defined for Kripke structures, on LTSs each state is assigned with the labels of the transitions that sprout out of the state. For a formal treatment of such interpretations see [31].

[11]Note that ϕ under the RCC constraint is expressible in CTL* [17], but with a fairness constraint in infinitary LTL, it would fall into FCTL [18] that can be checked more efficiently than CTL*.

[12]To be precise, *synth* corresponds to Paulson's *synth o analz* [32].

[13]Thinking of i and t as state properties that hold where the corresponding actions are executed, ϕ_A can be expressed in LTL as $\phi_A = G(i \Rightarrow F\ t)$.

4.3 Implementation

In our intruder $I^\dagger$, besides using the set X of messages, we also use another set Y, containing the messages that have been received but not yet sent by the intruder.[14] Another send action, $\mathbf{send}_I^\dagger$, for sending messages not belonging to Y, is used to detect and avoid any excessive (i.e. not required by RCC) collaboration by the intruder. The intruder $I^\dagger$ is modeled as

$$\begin{aligned} I^\dagger(X,Y) \;=\; & \\ & \sum_{m\in Msg,p,q\in P} \mathbf{recv}_I(p,\langle q,c\rangle).\\ & \quad I^\dagger(\{c\}\cup X, \{\langle q,c\rangle\}\cup Y) \;+\\ & \sum_{c\in Msg,p\in P} \mathbf{send}_I(p,\langle p,c\rangle).I^\dagger(X, Y\setminus\{\langle p,c\rangle\})\\ & \quad \lhd\langle p,c\rangle\in Y\rhd \delta \;+\\ & \sum_{c\in Msg,p\in P} \mathbf{send}_I^\dagger(p,\langle p,c\rangle).I^\dagger(X,Y)\\ & \quad \lhd synth(c,X)\wedge\langle p,c\rangle\notin Y\rhd\delta. \end{aligned}$$

The sets X and Msg, and the function $synth$ are the same as in section 4.2. The intruder starts with initial knowledge $X = K_0,\ Y = \emptyset$ and the protocol starts in its initial state P_0. We let $L_B \stackrel{def}{=} (S_B, s_{0B}, Act, Tr_B)$ be the LTS described by $\partial_{\Delta\cup\{\mathbf{send}_I^\dagger\}}(P(\mathsf{P}_0)\|I^\dagger(K_0,\emptyset))$.

As earlier, a fairness constraint is needed to restrict the behavior of the intruder. In particular, the intruder has to be fair when it comes to **send** actions and to sending messages (stored in set Y) that it received earlier but did not yet deliver to their intended recipient (see section 3). Implicitly, the intruder will also be fair on the protocol's internal actions, since it cannot control them. The only actions it does not need to treat in a fair way are the $\mathbf{recv}^\dagger$ actions, i.e. sending composed messages that it did not receive or that it did receive but delivered afterward. The fairness constraint can thus be expressed as "no possibilities, except for $\mathbf{recv}^\dagger$ actions, may be excluded forever". This constraint belongs to infinitary LTL and it can easily be combined with the property ϕ_A (in section 4.2) and expressed in the the regular alternation-free fragment of μ-calculus:

$$\phi_B \;\stackrel{def}{=}\; [\mathsf{T}^*\cdot i\cdot(\neg t)^*]\langle(\neg\mathbf{recv}^\dagger(\_))^*\cdot t\rangle\mathsf{T}$$

The intuitive meaning of ϕ_B is, whenever i has happened but t has not (yet) occurred, there is an execution path to t that does not contain any $\mathbf{recv}^\dagger$ actions. The intruder is thus not forced to collaborate more than delivering received messages. This means that those traces which contain $\mathbf{recv}^\dagger$ actions are either witnesses of attacks or excluded from successful runs. An *attack* in system B corresponds to $L_B \not\models \phi_B$.

4.4 Equivalence

The two systems, A and B, presented in sections 4.2 and 4.3 have different intruder models and use different fairness constraints. In system A the fairness constraint is a formalization of RCC, while for system B it is embedded in ϕ_B. It is far from obvious whether these systems are equivalent. The following theorem, which is the main result of this paper, states that when verifying a liveness property of the form ϕ then there is an attack in system A iff there is a corresponding attack in system B.

[14]For providing $RCC_n,\ n>1$, Y can be modeled as a multiset.

THEOREM 1. *For any protocol P,*

$$(P\|DY)\models_{RCC}\phi_A \;\equiv\; (P\|I^\dagger)\models\phi_B.$$

A proof sketch of this theorem is presented in appendix A.

When checking safety properties, one can ignore the ability of the DY intruder to do $\kappa.\delta$ actions (see section 4.2). The intuition behind this is that the intruder does not gain anything by quitting the protocol. This is obvious since safety attacks are finite traces. The intruder DY without $\kappa.\delta$ is thus equivalent to the intruder I, modulo renaming $\mathbf{send}_I^\dagger$ to $\mathbf{send}_I$ actions. Hence, the LTSs resulting from the interactions between an arbitrary protocol with these two intruder models are isomorphic under renaming. The following theorem articulates this observation. (the formalization and proof are omitted).

THEOREM 2. *If there is a safety attack for a protocol in system A, then there is a corresponding safety attack for the protocol in system B, and vice versa.*

4.5 Discussions

We end this section with a discussion of some features and limitations of the intruder model $I^\dagger$.

As is mentioned earlier, theorem 1 relies on the finiteness of the LTSs resulting from the interaction between a protocol and the intruder models. This poses some limitations: First, only a finite number of *actual* protocol sessions can be considered. To be precise, protocol participants cannot be provided with unbounded sources of fresh data. They can only run a finite number of protocol sessions and then either terminate or turn into puppet participants (as in [14]), which loop without generating new data. This restriction cannot be easily lifted in automatic verification techniques, since security of protocols in general is undecidable (see e.g. [15]). Second, if there are processes in the protocol with loops in their specifications, then the intruder cannot in general be provided with unbounded sources of new data (such as fresh nonces). In fact, trusted parties are often looping participants that perpetually resolve incoming requests in optimistic fair exchange protocols [4]. As another example, a vendor selling a finite number of items, that waits for a purchase request, handles it and recurs, can also be modeled as a looping participant. Whether the intruder can have access to an unbounded number of fresh data and still yield a finite state space, in the presence of looping participants, depends on the protocol being analyzed. In [12] it can, but not in [11].

Finally, we observe that usually only some of the channels in a communication network need to be resilient in order to achieve liveness. One way to implement this in system B is to add another set Z, characterizing messages which should be added to the buffer (set Y) of resilient channels. The only notable change in the specification of $I^\dagger(X,Y)$ would then be in the first sum (the receiving part) that would become (cf. section 4.3)

$$\begin{aligned} \sum_{c\in Msg,p,q\in P} & \mathbf{recv}_I(p,\langle q,c\rangle).I^\dagger(\{c\}\cup X,\{\langle q,c\rangle\}\cup Y)\\ & \lhd(p,\langle q,c\rangle)\in Z\rhd\\ & \mathbf{recv}_I(p,\langle q,c\rangle).I^\dagger(\{c\}\cup X,Y). \end{aligned}$$

The set Z does not necessarily need to specify contents of messages. Instead, the source or destination could be used, as it usually characterizes communications between honest participants and a trusted entity (see [12] for an example).

5. VERIFYING FAIR EXCHANGE PROTOCOLS

Assume a two-party (Alice and Bob) fair exchange protocol (multi-party protocols have similar requirements). When the protocol starts, both parties have an item. The purpose is to have Alice and Bob exchange their items such that the following properties hold [4]: First, if both parties behave according to the protocol and none of them aborts during the protocol round, then the protocol round terminates in a state where Alice has Bob's item and vice versa (*effectiveness*). Second, upon termination, either Alice has received Bob's item and Bob has received Alice's item, or none of the parties have lost their items (*fairness*). Third, each protocol round terminates for all parties that behave according to the protocol, and after the termination points the degree of achieved fairness will not change (*timeliness*). Note that timeliness contains a liveness property, namely the eventual termination of the protocol. To sum up, fair exchange requirements can be split into a safety part and a liveness part, and the safety requirements becomes meaningful only when the protocol is known to terminate.

A concern of a fair exchange protocol is to protect an honest party from possible malicious behavior of the other one. Therefore, when verifying fair exchange, the intruder is a legitimate, though malicious, principal of the protocol. Note that the fairness constraints introduced in sections 4.2 and 4.3 do not rule out any (malicious) behavior of the intruder as a protocol principal. In particular, the intruder can prematurely abort the protocol as a principal, while being forced to play the role of the network.

The termination requirement mentioned above can be expressed in the form of the liveness property ϕ (in section 4), where the actions i and t of an agent correspond to the engagement and termination, respectively, of a protocol session. Thus, according to theorems 1 and 2 both the safety and the liveness parts of fair exchange requirements can be checked in the proposed intruder model and inferred to hold in the DY model that respects the RCC_1 assumption:

COROLARY 1. *A fair exchange property holds in system A iff it holds in system B.*

6. RELATED WORK

We now review some notable work in formal analysis of fair exchange protocols and compare them to our work.

The liveness aspects of fair exchange are verified in [28, 23, 11]. In the former [28], a game-based semantics for fair exchange properties is presented, neatly expressing desired behaviors of fair exchange protocols. In [23] the SHVT toolset is used in verifying non-repudiation protocols. They reveal, in particular, a non-termination attack on the ZDB protocol [37]. In our terminology, both [28, 23] study the system $(P\|DY\|net)$, i.e. the intruder is separated from the network. What is gained is that enforcing RCC on *net* then simply boils down to applying the fairness constraint of definition 1. However, given the ability of the DY intruder to generate an infinite set of messages from any non-empty knowledge set, they need to constructively specify the set of messages passed in the protocol, as opposed to our recursive (similar to [32]) definition. Their intruder models are thus dependent on the protocol being analyzed and a predefined finite set of possible messages has to be manually anticipated. This in general defies the purpose of formal verification, that is to reason beyond human anticipation. Our work can be seen as extending the intruder model of [28] to a generic intruder.

The work in [11] reports an implementation of the intruder model $I^{\dagger}$ for analyzing a fair non-repudiation protocol.

In [26] a constraint solving approach is adopted to analyze finite rounds of contract signing protocols, while the intruder is provided with unbounded fresh data. Instead of resilient channels, they use synchronous authenticated secret channels, thus considerably restricting the intruder's abilities. Protocol participants are modeled as acyclic graphs in [26]. The liveness aspects of fair exchange requirements then naturally disappear and the resulting safety properties are shown to be decidable, under certain conditions. In contrast, our results (and also the results of [28]) are stated for an arbitrary protocol that can possibly contain looping participants, provided that the interaction between the protocol and the DY intruder results in a finite state space (see section 4.5 for more discussions). The analyses presented in [28, 23, 11] do not capture type flaw attacks[15]. Remarkably, these attacks are possible to detect in [26].

7. CONCLUSIONS

In this paper we studied some concurrency issues regarding the Dolev-Yao (DY) intruder model which arise when verifying liveness properties. A liveness property does not hold in general in this model without the resilient communication channels (RCC) assumption. We have given a formalization of RCC, whose complexity indicates that the standard DY intruder, obliged to respect RCC, is not suitable for automatic verification. We have thus proposed a modified DY intruder model that respects RCC and fits in the existing verification frameworks. The fact that the proposed intruder model can be implemented in a general purpose process algebra equips us with already well developed tools and techniques for modeling and verification.

Acknowledgments

We are grateful to Ana Almeida Matos, Ricardo Corin, Cas Cremers, Sandro Etalle, Wan Fokkink and Pieter Hartel for discussions and their helpful comments.

8. REFERENCES

[1] M. Abadi and B. Blanchet. Computer-assisted verification of a protocol for certified email. In *SAS '03*, volume 2694 of *LNCS*, pages 316–335, 2003.

[2] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996.

[3] B. Alpern and F. Schneider. Defining liveness. Technical Report TR 85-650, Dept. of Computer Science, Cornell University, October 1984.

[4] N. Asokan. *Fairness in electronic commerce*. PhD thesis, Univ. Waterloo, 1998.

[5] N. Asokan and P. Ginzboorg. Key agreement in ad hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.

[15] A type-flaw attack happens when a field in a message that was originally intended to have one type is interpreted as having another type. Type-flaw attacks have been shown to be easy to prevent [24].

[6] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *IEEE Symp. on Security and Privacy*. IEEE CS, 1998.

[7] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE J. Selected Areas in Communications*, 18(4):593–610, 2000.

[8] A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *WDAG '96*, volume 1151 of *LNCS*, pages 105–122. Springer, 1996.

[9] G. Bella and L. C. Paulson. Mechanical proofs about a non-repudiation protocol. In *TPHOL'01*, volume 2152 of *LNCS*, pages 91–104. Springer, 2001.

[10] J. Bergstra and J. Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37:77–121, 1985.

[11] J. Cederquist, R. Corin, and M. Torabi Dashti. On the quest for impartiality: Design and analysis of a fair non-repudiation protocol. In *ICICS'05*, volume 3783 of *LNCS*, pages 27 – 39. Springer, 2005.

[12] J. Cederquist and M. Dashti. Formal analysis of a fair payment protocol. In *Formal Aspect of Security and Trust*, volume 173 of *IFIP*, pages 41–54. Springer, 2004.

[13] I. Cervesato. The Dolev-Yao intruder is the most powerful attacker. In *LICS'01*. IEEE Computer Society Press, 16–19 June 2001.

[14] Y. Chevalier and L. Vigneron. Automated unbounded verification of security protocols. In *CAV '02*, volume 2404 of *LNCS*, pages 324–337. Springer, 2002.

[15] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *J. of Telecommunications and Information Technology*, 4:3–13, 2002.

[16] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, IT-29(2):198–208, 1983.

[17] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072. MIT Press, 1990.

[18] E. Emerson and C-L. Lei. Temporal reasoning under generalized fairness constraints. In *STACS '86*, pages 21–36. Springer-Verlag, 1985.

[19] R. Engelschall. URL rewriting engine. Apache HTTP Server Version 1.3.

[20] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[21] N. Francez. *Fairness*. Springer, 1986.

[22] J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In *Algebra of Communicating Processes*, Workshops in Computing Series, pages 26–62. Springer, 1995.

[23] S. Gürgens, C. Rudolph, and H. Vogt. On the security of fair non-repudiation protocols. In *ISC '03*, volume 2851 of *LNCS*, pages 193–207, 2003.

[24] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *J. Computer Security*, 11(2):217–244, 2003.

[25] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[26] D. Kähler and R. Küsters. Constraint solving for contract-signing protocols. In *CONCUR '05*, volume 3653 of *LNCS*, pages 233–247. Springer, 2005.

[27] S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, November 2002.

[28] S. Kremer and J. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *CONCUR'01*, volume 2154 of *LNCS*, pages 551–565. Springer, 2001.

[29] R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Sci. Comput. Program.*, 46(3):255–281, 2003.

[30] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE J. Selected Areas in Communication*, 21(2):44–54, 2003.

[31] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *J. ACM*, 42(2):458–487, 1995.

[32] L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6(1-2):85–128, 1998.

[33] A. Pnueli. a temporal logic of concurrent programs. *Theor. Comput. Sci.*, 13:45 – 60, 1981.

[34] V. Shmatikov and J. Mitchell. Finite-state analysis of two contract signing protocols. *Theor. Comput. Sci.*, 283(2):419–450, 2002.

[35] H. Vogt. Asynchronous optimistic fair exchange based on revocable items. In *Financial Cryptography*, volume 2742 of *LNCS*, pages 208–222. Springer, 2003.

[36] H. Vogt, H. Pagnia, and F. Gärtner. Using smart cards for fair exchange. In *WELCOM '01*, volume 2232 of *LNCS*, pages 101–113. Springer, 2001.

[37] J. Zhou, R. Deng, and F. Bao. Evolution of fair non-repudiation with TTP. In *ACISP '99*, volume 1587 of *LNCS*, pages 258–269. Springer, 1999.

APPENDIX

A. EQUIVALENCE PROOF SKETCH

THEOREM 1. *For any protocol P,*

$$(P\|DY) \models_{RCC} \phi_A \;\equiv\; (P\|I^\dagger) \models \phi_B.$$

Proof sketch: The idea is to give mappings between traces of L_A and L_B, such that the translation of an attack trace in any of these systems constitutes an attack in the other one. The translations that we give depend on the DY intruder's possibility (in system A) to perform κ and terminate (which it always can do, unless it already has done it), since the deadlock situation is not explicitly modeled in intruder I, while it is a behavior of DY. The set of states in which the DY intruder can perform κ is $S^\kappa = \{s \in S_A \mid \exists s' \in S_A.\ s \xrightarrow{\kappa} s'\}$. The notation $\mathcal{C}S^\kappa$ is used for the complement set of S^κ. In system A, the state of the intruder depends on set X and in system B it depends on the sets X and Y. We use the notations $\mathtt{X}(s)$ and $\mathtt{Y}(s)$ to refer to the states of the intruders and $\mathtt{P}(s)$ to refer to the state of the protocol P at state s.

For translating from L_B to L_A we define a function $F_S : S_B \to 2^{S_A}$ as

$$F_S(s) = \{\hat{s} \in S^\kappa \mid \mathtt{X}(\hat{s}) = \mathtt{X}(s) \wedge \mathtt{P}(\hat{s}) = \mathtt{P}(s)\}$$

and $F_{Act} : Act \to Act$ as

$$F_{Act}(a) = \mathbf{recv} \begin{cases} \mathbf{recv}, & \textit{if } a = \mathbf{recv}^\dagger \\ a, & \textit{otherwise.} \end{cases}$$

For the other direction, let $G = \langle G_S, G_{Act}\rangle$, where G_S and G_{Act} are the inverse images of F_S and F_{Act}, respectively. (From the definition of $I^\dagger(X, Y)$, it is obvious that $\forall a \in Act.\ |G_{Act}(a)| = 1$.) The function G_S is not total on S_A. So, we define another function $H = \langle H_S, H_{Act}\rangle$, where $H_S : \mathcal{C}S^\kappa \to 2^{S_B}$ is defined as

$$H_S(s) = \{\hat{s} \in S_B \mid \mathtt{P}(\hat{s}) = \mathtt{P}(s)\}$$

and $H_{Act} : Act \to Act$ is the identity function.

It turns out that, using the function F, traces in L_B can be uniquely translated into L_A. Conversely, using G and H, traces in L_A can be uniquely translated into L_B. One can think of F besides its inverses G and H as isomorphic translation functions which preserve transitions.

We need to show that ϕ_A holds for all traces that respect RCC in system A iff ϕ_B holds in system B. Suppose first that ϕ_A holds but not ϕ_B. Since ϕ_B does not hold, there is a trace α, containing i but no t after i, that cannot be extended to contain t without any $\mathbf{recv}^\dagger$ actions in between. Let α' be a translation of α to system A. System L_A is finite and finitely branching (see section 2.3), so α' can be extended in a fair way, let us say with β, such that $\alpha'.\beta$ respects RCC. Since ϕ_A holds, β must contain t. The translation of β back to system B now shows how α can be extended to attain t, which contradicts the assumption $\neg\phi_B$.

Now suppose that ϕ_B holds but not ϕ_A. Since $\neg\phi_A$, there is a trace α that respects RCC and contains i, but not t after i. The trace α is either finite or infinite. If α is finite then it is a deadlock trace and thus its translation to L_B violates ϕ_B. So, assume that α is infinite and let α' be the translation of α to L_B. There is a non-empty set of states $S' \subseteq S_B$ that α' visits an infinite number of times. Since ϕ_B holds, there is a state $s' \in S'$ that has an action $a \neq \mathbf{recv}^\dagger$ infinitely often enabled, but not taken by α'. From $a \neq \mathbf{recv}^\dagger$ it follows that $F_{Act}(a)$ infinitely often occurs in $\breve{\sigma}^\alpha$, but clearly is not taken by α. Thus α is not RCC-fair, contradicting the assumption. ∎