An Intruder Model for Verifying Termination in Security Protocols

Jan Cederquist
Department of Computer Science
University of Twente
The Netherlands
cederquistj@cs.utwente.nl

Muhammad Torabi Dashti
CWI
Amsterdam
The Netherlands
dashti@cwi.nl

Abstract

We formally describe an intruder that is suitable for checking fairness properties of security protocols. The intruder is proved to be equivalent to the Dolev-Yao intruder that respects the resilient communication channels assumption, in the sense that, if a fairness property holds in one of these models, it also holds in the other.

Keywords: *Fairness properties, Formal verification, Dolev-Yao intruder, Resilient communication channels assumption, Security protocols.*

1 Introduction

Traditionally security is defined in terms of safety properties [16], expressing that some undesirable states are never reached. However the growth of electronic commerce applications and their new requirements, like fair exchange [17], makes liveness an indispensable part of the definition of security. In contrast to safety, liveness guarantees that a certain state will be reached. As is mentioned in [15], one of the current issues in the formal analysis of cryptographic protocol is that the standard model of intruder (known as the Dolev-Yao model [7]) is not suitable for verification of liveness in the context of security protocols. The Dolev-Yao intruder has, in principle, complete control over the communication channels and no liveness property can be proved for a protocol running in such an environment. Instead, when proving liveness properties, it is usually assumed that the communication channels are resilient, i.e. all transmitted messages will eventually reach their destination. It would thus be desirable to verify liveness in the presence of a (modified) Dolev-Yao intruder that respects this assumption.

Compared to safety properties, liveness properties are also considerably harder to prove and sometimes formal verification of fair exchange or non-repudiation protocols, which require liveness, leave it out or let it be based on informal arguments. As a matter of fact, there are known liveness attacks [19], which show the importance of extending the realm of formal verification to the liveness security requirements. There are however a few interesting approaches to automated formal verification of security protocols where liveness is also verified. A game-based analysis of fair exchange and non-repudiation is presented by Kremer and Raskin in [12]. Their method provides a neat way to

express the desired properties. However, their model of intruder is protocol specific and cannot be easily extended to large protocols. Gürgens and Rudolph [10] present an approach using asynchronous product automata for verification of fair exchange and non-repudiation protocols. A Dolev-Yao intruder that respects the Resilient Communication Channels assumption (RCC) was described and implemented, for some special cases, in [4].

In this paper, we formalize RCC and we present a process-algebraic model of intruder as well as the pattern of properties for checking fairness. Our suggested intruder model is generic (not dependent of a particular protocol). For fairness properties, it is equivalent to the Dolev-Yao intruder which respects RCC. In this paper we sketch the proof of this fact. In contrast to [10], we use a synchronous model. This simplifies description and implementation of the intruder, since no further restrictions are needed to prevent generation of messages that no one can receive. Furthermore, our intruder has been implemented and used for verifying fairness for non-repudiation protocols [5].

The rest of the paper is organized as follows. In section 2 we give some definitions and introduce notations used later. In section 3 we formalize the notion of resilient communication channels. Our intruder is defined in section 4. There we also sketch the proof of the fact that our intruder is equivalent to the Dolev-Yao intruder that obeys the resilient communication channels assumption. Some concluding remarks are given in section 5.

2 Preliminaries

In this section we briefly recall the notions of safety, liveness and fairness properties. The communication model we are using is described here. We also introduce notations for sequences and labeled transition systems.

2.1 Verifying Liveness Properties

Properties of systems are usually divided into two main classes [13]: *safety* (something bad will never happen) and *liveness* (something good will eventually happen).

Safety properties are often verified in the Dolev-Yao model [7]. The Dolev-Yao intruder has complete control over the network. It remembers all messages that have been transmitted. It can decrypt and sign messages, if it knows the corresponding key. It can compose and send new messages from its knowledge. It can also remove or delay messages in favor of others being communicated. The Dolev-Yao intruder has been shown to be the most powerful intruder for safety properties [6]. Liveness properties, however, do not hold in the Dolev-Yao model, since the intruder can simply stop all communications before the desired event has occurred. So, when verifying liveness properties one often needs to assume the communication channels to be resilient, meaning that messages sent over the network will eventually be delivered. For a liveness property to hold, the Dolev-Yao intruder is thus required to cooperate to some extent.

2.2 Fair Exchange Properties

In a fair exchange protocol, there are two (or more) parties A and B. When the protocol starts, both of them have an item. The purpose is to have A and B exchange their items, in a "fair" manner. The primary requirements for a fair exchange protocol is *effectiveness*, *fairness* and *timeliness* (cf. [1]). Effectiveness means that if both parties behave according to the protocol and none of them wants to abort during the protocol round, the protocol round will terminate in a state where A has B's item and vice versa. Effectiveness which is verified without intruder is omitted from the rest of the paper. An exchange protocol is called fair if, when it has terminated, either A has received B's item and B has received A's item, or none of the parties have lost their items. Timeliness means that protocol rounds will terminate for all parties (that behave according to the protocol) and after the termination points the degree of achieved fairness will not change.

According to the above, once we have convinced ourselves that a system terminates, fairness and timeliness can be verified as safety properties (see also [20]). Therefore termination is the only liveness property that needs to be checked.

2.3 Communication Model

We consider a set E of deterministic processes (agents) that communicate over a network. For sending and receiving messages m the agents perform the actions $send(m)$ and $recv(m)$. Even though an agent A sends the message m with the intention that it should be received by agent B performing $recv(m)$, in fact it is the network that receives the message from A, and it is from the network that B can receive m. In this way the network can be seen as an external agent (not in E).

The sending and receiving of messages between agents and the network are synchronized, an agent A can only send a message m intended for B if the network at the same time receives it from A and B can only receive a message m (believingly) from A if the network at the same time sends it to B. This synchronization is denoted com. We use the notation com_{send} for the action that represents the synchronization between the network and E, when the network performs a $send$ action. The synchronization com_{recv} is defined correspondingly. The names of sender and (intended) recipient of a message are assumed to be part of the message itself.

To model an intruder that has complete control over the network, we assume it plays the role of the net. In fact, in this paper, the intruder *is* the network. Below the notation for the network and intruder will be used interchangeably. Since we focus on the intruder, the set of (honest) agents E is called the environment (of intruder). For generality, the environment can perform internal actions, not controlled by the intruder, such as messages sent over secure channels.

The main benefit of using a synchronous model is that it simplifies the modeling of the intruder. Indeed, in a synchronous communication between agents and the network, even a very powerful intruder (like the Dolev-Yao intruder) with capabilities to generate lots of messages, will only generate messages that other agents can receive.

2.4 Sequences and Sequences of Ready Sets

We will deal with sequences

$$a : I \to X,$$

where the index set I is either $\{0, \ldots, n-1\}$ (for finite sequences of length n) or the natural numbers Nat (for infinite sequences) and X is some finite set. Sequences of *ready sets*[1]

$$A : I \to \mathcal{P}(X),$$

where $\mathcal{P}$ is the power set, are used for describing possibilities. Intuitively, the ready set A_i contains the possibilities after $a_0, \ldots, a_{i-1}$ have happened.

2.5 Labeled Transition Systems

We use labeled transition systems for describing executions of security protocols:

Definition 1 *A Labeled Transition System (LTS) is a tuple (S, s_0, Lab, Tr), where S is a finite set of states, $s_0 \in S$ is the initial state, Lab is a set of labels and $Tr \subseteq S \times Lab \times S$ is a 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 performing a.*

The labeled transition systems are assumed to be deterministic, i.e. for each state $s \in S$ and each action $a \in Act$, there is at most one state $s' \in S$ such that $s \xrightarrow{a} s'$. This is a quite reasonable assumption since the labeled transition systems discussed here origin from security protocols. The labels in the transition systems in this paper are actions belonging to a finite set Act and, the notation $s_0 \overset{A}{\rightsquigarrow} s_n$ will be used to express that state s_n is reachable from state s_0 using only actions in $A \subseteq Act$, i.e. there is a sequence a (possibly empty) such that $a_i \in A$ $(i < n)$ and $s_0 \xrightarrow{a_0} \cdots \xrightarrow{a_{n-1}} s_n$. Sequences of actions will also be referred to as traces.

3 The Resilient Communication Channels Assumption

The aim of this section is to formalize the Resilient Communication Channels assumption (RCC), saying that all messages transmitted over the network will eventually be delivered.

In section 3.3, RCC is defined as a fairness constraint[2], for a certain labeled transition system, using the definition of fair traces in section 3.2.

3.1 Finiteness

For RCC, messages that have been sent to the network (intruder), but not yet delivered, need to be stored. However, in order to make sure that the state space remaines finite, only a finite number of each such message can be stored.

[1] The vocabulary *ready set* is borrowed from the notion of *ready traces*, cf. [2].

[2] Two notions of fairness are used in this paper; fairness of an exchange protocol (section 2.2) and fairness constraints of a labeled transition system. The second one is used to describe *fair* execution traces. Then, when verifying a liveness property, only the fair execution traces are considered.

Definition 2 *RCC_n means that the network can store at most n instances of each message, and all received messages will eventually be delivered.*

Assuming RCC_n and a finite number of protocol sessions and players, and assuming that type flaw attacks are impossible[3], then the state space will remain finite. This restriction on RCC is thus, in general, needed for the state space generation to terminate. Also, by assuming RCC_n (for some n), we are on the safe side. If a liveness property holds assuming RCC_n, it also holds for RCC in general.

Lemma 1 *If a certain liveness property holds with respect to RCC_n, the property also holds with respect to RCC_{n+1} (and RCC in general).*

Choosing a small number n when assuming RCC_n is of course important for the size of the state space. By assuming RCC_n, if one action is used for different things in a protocol, the analysis may in the worst case result in false attacks. In the formalization of RCC (in section 3.3) only *one* instance of each sent message is required to be delivered. For instance, after the sequence $com_{recv}(m).com_{recv}(m)$ of actions, only one $com_{send}(m)$ action is required (if the environment can receive m), not two. From this point on, we assume RCC_1 when we talk about RCC.

3.2 Fair Traces

Here the aim is to formalize a notion of fairness constraints for labeled transition systems, that informally says that *no possibilities are excluded forever.* We propose the following definition of fairness for sequences with respect to sequences of ready sets:

Definition 3 *Let $a : Nat \to X$ be a sequence of elements of X and $A : Nat \to \mathcal{P}(X)$ a sequence of ready sets. We say that a is* fair *with respect to A iff, for all $x \in X$, if $\{i \in Nat \mid x \in A_i\}$ is infinite, then so is $\{i \in Nat \mid x = a_i\}$.*

Note that finite sequences are trivially fair. Some more specialized fairness constraints are defined in section 3.3.

Definition 4 *A sequence a of actions belongs to a LTS (S, s_0, Act, Tr) iff there is a sequence s of states, starting with the initial state, such that $\forall i.(s_i, a_i, s_{i+1}) \in Tr$.*

If a sequence a of actions belongs to a LTS L, the corresponding sequence of states (see definition 4) is unique. This follows easily by induction, using the fact that L is deterministic (which was assumed in section 2.5). The sequence a also defines a sequence of transitions and a sequence of ready sets of transitions:

Definition 5 *Let a be a sequence of actions belonging to a LTS (S, s_0, Act, Tr) and let s be the corresponding sequence of states. An associated sequence of transitions and a sequence of ready sets of transitions can be defined as*

$$t_i = (s_i, a_i, s_{i+1})$$

and

$$T_i = \{(s_i, \alpha, s') \in Tr \mid \alpha \in Act, s' \in S\}.$$

[3]Type flaw attacks can be prevented (see [11]) and here they are assumed not to occur.

We can now define when a ready trace is fair in a LTS:

Definition 6 *Let a be a trace that belongs to a LTS L, t and T the corresponding sequences of transitions and ready sets of transitions, respectively. The trace a is fair in L iff t is fair with respect to T.*

The reason for translating the trace a to a sequence of transitions and a sequence of ready sets of transitions (instead of using a sequence of ready sets of actions), in definition 6, is to more easily avoid certain traces to be fair. Consider the infinite trace $a = \alpha \cdot \beta \cdot \alpha \cdot \beta \cdot \cdots$ in the following LTS:

$$(\{s_0, s_1, s_2\}, s_0, \{\alpha, \beta\}, \{s_0 \xrightarrow{\alpha} s_1\, s_1 \xrightarrow{\beta} s_0\, s_0 \xrightarrow{\beta} s_2\}).$$

We do not want a to be fair since, a visits s_0 infinitely often but, from there it never performs β and moves to s_2.

3.3 Formalization of RCC

In section 4 we will study labeled transition systems which result from interactions between intruders and an environment. Each state s of such a LTS is thus the product of the state of the intruder I and the state of the environment E. We use the notations $\mathsf{X}(s)$ and $\mathsf{E}(s)$ for the corresponding state of I and E, respectively, and the notation $\mathsf{E}(s) \models \alpha$ means that, at state s, some agent in E can perform the action α.

In the formalization of RCC below, the following definitions of messages that the environment can receive/send from/to the network come useful:

Definition 7 *The set of messages that the environment E can receive from the network (intruder) at state s:*

$$r(s) = \{m \in message \mid \mathsf{E}(s) \models com_{recv}(m)\}.$$

The set of messages that E can receive if it is allowed to first perform an arbitrary number of internal actions

$$r^*(s) = \{m \in message \mid \exists s' \in S.s \overset{A_E}{\rightsquigarrow} s' \wedge \mathsf{E}(s') \models com_{recv}(m)\},$$

where A_E represents the internal actions of E. Similarly, we define the set of messages that E, in state s, can send

$$g(s) = \{m \in message \mid \mathsf{E}(s) \models com_{send}(m)\}$$

and, if E is allowed to first perform an arbitrary number of internal actions,

$$g^*(s) = \{m \in message \mid \exists s' \in S.s \overset{A_E}{\rightsquigarrow} s' \wedge \mathsf{E}(s') \models com_{send}(m)\}.$$

Now we will try to capture the notion Resilient Communication Channels assumption (RCC) and define formally what it means for a network (intruder) respects RCC. We do this by defining a fairness constraint that says when a trace respect RCC. First, we specify the (resilient) actions that eventually must occur. They consist of the intruder's send actions (for messages it has received but not yet sent), actions that are internal to the environment and the intruder's receive actions:

Definition 8 *Let a be a sequence of actions belonging to a LTS, and let T be the corresponding sequence of ready sets of transitions (according to definition 5). The sequence of sets of* resilient *transitions (σT) is defined as*

$$\begin{aligned}(\sigma T)_i \;=\; & \{(s_i, com_{send}(m), s') \in T_i \mid \\ & \quad \exists j < i. a_j = com_{recv}(m) \;\wedge \forall k.\; j < k < i. a_k \neq com_{send}(m)\} \bigcup \\ & \{(s_i, x, s') \in T_i \mid x \textit{ is internal in } E\} \bigcup \\ & \{(s_i, com_{recv}(m), s') \in T_i\}.\end{aligned}$$

As mentioned before, the messages are assumed to contain (original) sender and (intended) recipient. This guarantees that messages eventually are delivered to the "right" recipient.

Definition 9 *Let a be a sequence of actions belonging to a LTS L. Let T be the corresponding sequence of ready sets of transitions (according to definition 5). The trace a is fair in L, according to RCC (denoted RCC-fair) iff a is fair with respect to (σT) (see definition 6).*

In addition to RCC-fairness, we also specify under what circumstances the intruder should be allowed to stop collaborating. Again this is defined as a fairness constraint. The intruder should be allowed to stop if there is nothing to receive from the environment, and all messages (to which there is an recipient in the environment), that it has received earlier, have been sent:

Definition 10 *Given a LTS. A trace a is said to* respect *RCC iff it is RCC−fair and*

$$\begin{aligned}\forall i.\; a_i = stop \;\Rightarrow\; & g^*(s_i) = \emptyset \wedge \\ & \forall m \in r^*(s_i). \forall j < i.\; a_j = com_{recv}(m) \;\Rightarrow\; \exists k. j < k < i \wedge \\ & \qquad\qquad a_k = com_{send}(m),\end{aligned}$$

where stop is a particular action the intruder performs when it intends to stop collaborating.

4 An Intruder for Verifying Fairness Properties

Here two systems are described, each one defining an intruder that communicates with an environment of (honest) agents. The intruders in the two systems are specified by pseudo code in a process algebraic language. The environment in the two systems is arbitrary, only its actions are specified. For each system we also describe how the intruders communicate with the environment and what constitutes an attack.

In the first system (section 4.2) we have the Dolev-Yao intruder and a formalization of the resilient communication channels assumption (RCC), as a fairness constraint. We are aiming at automatic verification of fairness properties and from that point of view there are some problems with the first system. First, the RCC constraint is a quite complicated formula. In model checking, the verification time is exponential in the length of the formulas expressing fairness constraints and properties to be verified. Second, the RCC constraint is not generic, it is protocol specific. In the second system,

the implementation, we present another intruder, that is more suitable for automatic verification of fairness properties. The two systems are proved to be equivalent with respect to fairness properties[4], in the sense that if one of them is insecure (an attack can be found against a certain fairness property), then so is the other one.

4.1 Process algebra

In the pseudo codes below, the symbols '.' and '+' are used for the sequential and alternative composition operator, respectively. The operator $\sum_{d\in D} P(d)$ behaves like $P(d_1) + P(d_2) + \cdots$. The constant δ expresses that, from now on, no action can be performed. As mentioned earlier send and receive actions are synchronized. The intruders can only receive/send a message if an agent in the environment at the same time sends/receives it, and vice versa. The intruders' knowledge is represented by the sets of messages X and Y. The set-operations have their usual meaning. Finally, using the predicate $synth(m, X)$[5], the intruder checks whether it can synthesize the message m from its knowledge X. In this check it is assumed that the intruder can compose and de-compose messages and, in particular, decrypt and sign messages if it knows the corresponding key.

4.2 Specification

The Dolev-Yao intruder can receive messages, which are added to its knowledge. It is able to send messages that can be synthesized from its knowledge. It can also terminate (it is assumed to perform a certain action *stop* immediately before termination). The pseudo-code for such an intruder can be given by

$$\begin{aligned} DY(X) = \quad & \sum recv(m).DY(X \cup \{m\}) + \\ & \sum_{synth(m,X)} send(m).DY(X) + \\ & stop.\delta \end{aligned}$$

As mentioned earlier, messages between agents in the environment are sent via the intruder (unless they are internal) and names of the intended recipient of a message is assumed to be part of the message itself.

The environment has the actions *init*, *terminate*, *send*, *recv* and internal actions that the intruder has no control over. Send (*send*) and receive (*recv*) actions are synchronized between intruder and environment and we use the notation com_z for the synchronization when the intruder performs an action z. Moreover, we assume the intruder starts with the initial knowledge $X = K_0$ and the environment starts in the initial state E_0. The system described above is called system A, and the resulting labeled transition system is denoted L_A.

To express the termination property, the actions *init* and *terminate* are used

$$P_A(a) \;=\; \forall i.a_i = init \Rightarrow \exists j > i.a_j = terminate,$$

[4] We sketch the proof of this fact in section 4.4.

[5] Cf. the *synth* operator in [18].

i.e. for each *init* action there is a *terminate* action coming later.

The intruder DY is the Dolev-Yao intruder. As mentioned in the introduction (section 1), the intruder DY must be restricted in order to allow the verication of liveness properties. So, only the traces in L_A that respect RCC are considered. Thus, there is an attack in system A if there is a trace a in L_A that respects RCC and violates P_A.

4.3 Implementation

A Dolev-Yao intruder that is suitable for liveness properties should have the ability to compose messages from its knowledge and respect the resilient communications channels assumption. To implement that behavior, we use another set of knowledge Y (besides X), containing the messages that have been received but not yet sent. Another send action $send^\dagger$ for messages taken from Y is also used:

$$\begin{aligned} I(X,Y) = \quad & \sum recv(m).I(X \cup \{m\}, Y \cup \{m\}) + \\ & \sum_{m \in Y} send(m).I(X, Y \setminus \{m\}) + \\ & \sum_{synth(m,X) \wedge m \notin Y} send^\dagger(m).I(X,Y) \end{aligned}$$

The environment is the same as for the specification (section 4.2). The actions $send(m)$ and $recv(m)$ are synchronized between intruder and environment, as well as $send^\dagger(m)$ and $recv(m)$. Again, the notation com_z is used to represent the synchronization point where the intruder performs an action z. We assume the intruder starts with the initial knowledge $X = K_0$, $Y = \emptyset$ and the environment starts in the initial state E_0. The system described above is called system B, and the resulting labeled transition system is denoted L_B.

As was the case with system A, because of loops, a fairness constraint is needed to restrict the behavior of the intruder. But in system B, the fairness constraint is easier to describe. The intruder has to be fair when it comes to receive actions and in sending messages that it earlier received. (Implicitly, the intruder will also be fair on the environment's internal actions, since it cannot control them.) The only actions it does not need to treat in a fair way are the $send^\dagger$-actions, i.e. sending composed messages that was not received. The termination property can thus be expressed as, whenever the action *init* has happened, but not (yet) *terminate* has occurred, there is a "path" to *terminate* that does not contain any $com_{send^\dagger}$ actions. We define this property in μ-calculus[6]:

$$P_B \;=\; [T^*.init.(\neg terminate)^*] < (\neg com_{send^\dagger})^*.terminate > T.$$

Note that the fairness constraint is built-in in this property.

We say there is an attack in system B if this property is violated, i.e. there is a trace in L_B that contains *init*, but no *terminate* (after *init*), that cannot be extended to contain *terminate* without any $com^\dagger_{send}$-actions.

[6]We use '.', '¬' and '*' for concatenation, complement and reflexive-transitive closure for regular formulas, respectively. In action formulas, T means *any action*, and in state formulas, it represents the entire state space. The operators $\langle \ldots \rangle$ and $[\ldots]$ have their usual meaning ($\Box$ and $\Diamond$ in modal logic).

4.4 Equivalence

In the two previous sections we have defined the properties P_A and P_B, which basically say that, if a certain action *init* happens, another action *terminate* will eventually happen.

Theorem 1, below, says that, P_A holds for all traces in system A that respect RCC iff P_B holds for all traces in system B. Since fair exchange properties can be split up into termination and safety properties (section 2.2), it follows that the two intruder models are equivalent with respect to fairness properties. (For safety properties, the Dolev-Yao intruder can be used in both systems.)

Theorem 1 *If there is an attack for system A, there is a corresponding attack for system B, and vice versa.*

Proof sketch: We will first define functions for translating traces between the systems $L_A = (S_A, s_{0_A}, Act_A, Tr_A)$ and $L_B = (S_B, s_{0_B}, Act_B, Tr_B)$.

In section 3.3, we introduced the notations $\mathsf{X}(s)$ and $\mathsf{E}(s)$ to refer to the states of the intruder and environment in state $s \in S_A$, respectively. In system B, the state of the intruder depends on two sets, X and Y, and we use the notations $\mathsf{X}(s)$ and $\mathsf{Y}(s)$ to refer to the states of the intruder at state $s \in S_B$.

Take a function $F : S_B \to S_A$ such that

$$\mathsf{X}(F(s)) = \mathsf{X}(s) \ \wedge \ \mathsf{E}(F(s)) = \mathsf{E}(s).$$

Also, define $F : Act_B \to Act_A$ (the name F is overloaded) as

$$F(\alpha) = \begin{cases} com_{send(m)}, & \text{if } \alpha = com_{send^\dagger(m)} \\ \alpha, & \text{otherwise.} \end{cases}$$

F can thus be used to map transitions. It can be shown that such an $F : Tr_B \to Tr_A$ is a homomorphism, it preserves transitions.

Let G be the inverse image of F. F is not surjective, G is thus not total on Tr_A. So, we define a relation H on $Tr_A \times Tr_B$ by (H is also overloaded)

$$H(s, s') \text{ iff } \mathsf{E}(s) = \mathsf{E}(s'),$$

for $s \in S_A \setminus Im(F)$ (where $Im(F)$ is the image of F), and

$$H(a, a),$$

for actions a. It turns out that F, G and H translate traces uniquely between the systems A and B.

We need to show that, if P_A holds (for all traces that respect RCC) in system A, P_B holds (for all traces) in system B, and vice versa. Suppose first that P_A holds but not P_B. Since P_B does not hold, there is a trace a, containing *init* but not *terminate* (after *init*), that cannot be extended to contain *terminate* with only $\neg com_{send^\dagger}$ actions. Let a' be the translation of a. L_A is finite. So a' can be extended fair way, lets say with b, such that $a'.b$ respects RCC. Since P_A holds b must contain *terminate*. The translation

of b to system B now shows how a can be extended to a trace that contain *terminate*, which contradicts the assumption $\neg P_B$.

Now, suppose $\neg P_A$ and P_B. Since $\neg P_A$, there is a trace a that respects RCC and contains *init* but not *terminate* (after *init*). Let a' be the translation of a to system B. It can be shown that, if a is finite (ends in a deadlock), a' violates P_B. So, let us assume a is infinite. Let $S' \subseteq S_B$ be the (non-empty) set of states that a' visits an infinite number of times. Since P_B holds, from each state in S', there is a path which reaches *terminate* without $com_{send^\dagger}$ actions. Consider one of these paths, let s' be the state where it leaves a', α be the first action outside a' and s'' the next state. The transition (s', α, s'') is infinitely often possible to follow, but never taken by a'. Also, $F(s', \alpha, s'')$ is infinitely often possible by a, but never taken. Moreover, since α is not a $com_{send^\dagger}$ action, $F(s', \alpha, s'')$ belongs to $(\sigma T)_i$, where T is the sequence of ready sets of transitions associated to a (definition 5). Thus, $F(s', \alpha, s'')$ belongs to an infinite number of sets $(\sigma T)_j$. But, it is never taken by a. This contradicts the fact that a is RCC-fair.

5 Conclusion

We have formalized the resilient communication channels assumption. We have also given an implementation of a generic intruder that is suitable for verifying fairness properties. We have shown that a system consisting of an environment of (honest) agents and our intruder reaches termination iff, the same system agents and the Dolev-Yao intruder that respects RCC terminates. Our intruder has also been implemented [5] in the process algebraic language μCRL [9, 3] and used for verifying fairness for non-repudiation protocols using EVALUATOR 3.0 [14] from the CADP tool-set[8].

When it comes to safety properties, our intruder model (and also the Dolev-Yao intruder that respects RCC) is equivalent to the Dolev-Yao intruder. This is almost trivial since safety attacks are finite traces. However, for safety properties, the intruder needs to be extended so that it "tries to reach" a bad state (for instance, using $synth(m, X)$, it checks whether it can synthesize a secret m from gathered knowledge X).

We cannot say that the intruder presented here is equivalent to the Dolev-Yao intruder, that respects RCC, for liveness properties in general. However, for security protocols, one can argue that termination is the most important liveness property, and often termination is the only liveness property that needs to be verified. If the protocol sessions are not intended to interfere with each other, any property can be reduced to termination and two safety properties (in the same way as in section 2.2): (1) a certain property P holds at the termination point and, (2) after the termination point, events affecting P do not occur. Here only termination is a liveness property.

References

[1] N. Asokan. *Fairness in electronic commerce*. PhD thesis, University of Waterloo, 1998.

[2] J. A. Bergstra, J. W. Klop, and E.-R. Olderrog. Readies and failures in the algebra of communicating processes. *SIAM J. Comput.*, 17(6):1134–1177, 1988.

[3] S. Blom, W. Fokkink, J. F. Groote, I. van Langevelde, B. Lisser, and J. van de Pol. μCRL: A toolset for analysing algebraic specifications. In *Proceedings of the 13th International Con-*

ference on Computer Aided Verification, volume 2102 of *LNCS*, pages 250–254. Springer-Verlag, 2001.

[4] J. Cederquist and M. T. Dashti. Formal analysis of a fair payment protocol. In *Formal Aspect of Security and Trust*, volume 173 of *IFIP*, page to appear, Toulouse, France, 2004. Springer-Verlag.

[5] J. G. Cederquist, R. J. Corin, and M. T. Dashti. On the quest for impartiality: Design and analysis of a fair exchange protocol (draft). Unpublished manuscript, `http://www.cs.utwente.nl/~cederquistj/ccd.pdf`, 2005.

[6] I. Cervesato. The Dolev-Yao Intruder is the Most Powerful Attacker. In J. Halpern, editor, *16th Annual Symposium on Logic in Computer Science — LICS'01*, Boston, MA, 16–19 June 2001. IEEE Computer Society Press.

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

[8] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A protocol validation and verification toolbox. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification*, volume 1102 of *LNCS*, pages 437–440. Springer-Verlag, 1996.

[9] J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, editors, *Algebra of Communicating Processes '94*, Workshops in Computing Series, pages 26–62. SV, 1995.

[10] S. Gürgens and C. Rudolph. Security analysis of (un-) fair non-repudiation protocols. In *FASec*, pages 97–114, 2002.

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

[12] S. Kremer and J. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *Proceedings of the 12th International Conference on Concurrency Theory*, volume 2154 of *LNCS*, pages 551–565. Springer-Verlag, 2001.

[13] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.

[14] R. Mateescu. Efficient diagnostic generation for boolean equation systems. In *Proceedings of 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2000*, volume 1785 of *LNCS*, pages 251–265. Springer-Verlag, March 2000.

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

[16] J. K. Millen. Applications of term rewriting to cryptographic protocol analysis overview of the invited talk. *Electr. Notes Theor. Comput. Sci.*, 36, 2000.

[17] H. Pagnia, H. Vogt, and F. C. Gärtner. Fair exchange. *The Computer Journal*, 46(1):55–7, 2003.

[18] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Comput. Secur.*, 6(1-2):85–128, 1998.

[19] C. Rudolph S. Gurgens and H. Vogt. On the security of fair non-repudiation protocols. In *Proc. of Information Security Conference*, volume 2851 of *LNCS*, pages 193–207. Springer-Verlag, 2003.

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