A Game-Based Verification of Non-Repudiation and Fair Exchange Protocols*

Steve Kremer Jean-François Raskin
Département d'Informatique
Faculté des Sciences
Université Libre de Bruxelles, Belgium
Steve.Kremer@ulb.ac.be Jean-Francois.Raskin@ulb.ac.be

Abstract

In this paper, we report on a recent work for the verification of non-repudiation protocols. We propose a verification method based on the idea that non-repudiation protocols are best modeled as games. To formalize this idea, we use alternating transition systems, a game based model, to model protocols and alternating temporal logic, a game based logic, to express requirements that the protocols must ensure. This method is automated by using the model-checker MOCHA, a model-checker that supports the alternating transition systems and the alternating temporal logic. Several optimistic protocols are analyzed using MOCHA.

1 Introduction

Non-repudiation protocols. During the last decade, open networks, above all the Internet, have known an impressive growth. As a consequence, new security issues, like non-repudiation have to be considered. Repudiation is defined as the *denial* of an entity of having participated in all or part of a communication. Consider for instance the following scenario: Alice wants to send a message to Bob; after having sent the message, Alice may deny having sent it (repudiation of origin), or Bob may deny having received it (repudiation of receipt). Therefore specific protocols have been designed, generating both a non-repudiation of origin (NRO) evidence, destined to Bob, and a non-repudiation of receipt (NRR) evidence, intended to Alice. These evidences are based on digital signatures that provide proofs of the origin of a message or a receipt. In case of a dispute Alice or Bob can present their evidences to an adjudicator, who can take a decision in favor of one of the two entities without ambiguity. One of the major problems in these protocols arises when we want to achieve fairness, i.e. avoid that one of the entities gets its evidence without the other one being able to also get his evidence. We have to handle the fact that at one moment during the protocol execution, an entity will come into an advantageous position, by being the first one to receive the expected evidence. For instance, if Alice starts sending her message with her non-repudiation of origin evidence, Bob has received all expected information and may stop the protocol being in an advantageous position.

In 1980 Even and Yakobi [11] have shown that there is no deterministic protocol (without trusted third party) which solves the contract signing problem (contract signing is a closely related

*Parts of this works have previously been presented in [14].

problem, where Alice and Bob aim to exchange digital signatures on a contract in a fair way). Therefore two different approaches have been considered. A first one is without trusted third party (TTP for short) intervention, while the second one needs a TTP.

Protocols without TTP are either based on gradual exchange or are probabilistic protocols. The disadvantage of protocols based on gradual release [26, 27] is that they generally require that all involved parties have the same or a related computing power. Moreover the number of messages that are needed to realize the exchange is rather high, and makes this approach inefficient. In [6] a probabilistic protocol for digital contract signing is proposed: the probability that the contract has been signed, is increased each round until reaching one. The assumption of same computational power is not needed. While this protocol needs the intervention of a judge, in a similar way to the TTP intervention in optimistic protocols, i.e. the judge is only needed when the protocol does not terminate correctly, another recently presented probabilistic non-repudiation protocol [18] succeeded in relaxing the condition on the computational power, without any TTP intervention. Here the idea is that the recipient does not know a priori which transmission will contain the message. The probability of guessing the transmission including the message is arbitrarily small. However to decrease the probability the number of messages has to be increased. Recently, a protocol based on the idea of gradual exchange [7] also succeeded to relax the assumption of equivalent computational power, by using an underlying problem that is hard to parallelize. However, the protocol confirms the believe that solutions without TTP require a lot of transmissions.

The second approach is the one using a TTP. Here, a possible scenario is to first send the message to the TTP, who acts as an intermediary to assure delivery. The major problem of this approach is the network and communication bottleneck, created at the TTP. To avoid the performance decrease created by this bottleneck, in 1997, Asokan et al. and Micali independently introduced the optimistic approach for fair exchange in [3] and [20]. In an optimistic protocol one supposes that in general the involved entities are honest and the network is well functioning. The rationale is that the TTP only intervenes in case of a problem (a cheating entity or a network failing a delivery at a critical moment). Afterwords Zhou et al. applied the optimistic approach to the non-repudiation protocols [30]. These protocols are becoming rather complex as they require that at some moment in the protocol, Alice and Bob, are able to contact the TTP to either force a successful ending, or to make a later successfully ending impossible. Optimistic protocols are the ones that received most of the attention in recent literature.

A non-repudiation protocol has to ensure several properties. The first one is *fairness*[1]: fairness must ensure that if at least one entity is honest, either both entities receive the expected non-repudiation evidence or none of them receives it. This property can also be split in the following two properties. *Fairness for Alice* ensures that, if Alice is honest, Bob receives the non-repudiation of origin evidence only if Alice receives her non-repudiation of receipt evidence. *Fairness for Bob* ensures that, if Bob is honest, Alice receives her non-repudiation of receipt evidence only if Bob receives his non-repudiation of origin evidence. The fairness requirement is the conjunction of these two properties. Another property we require is *timeliness*: we want that the protocol finishes for each honest player after a finite amount of time. A third property that is desirable but not necessary is viability. A protocol is *viable* if two honest players always succeed in exchanging the expected evidences. In general viability can only be realized by strengthening the requirements we make on the communication channels. We define three classes of channels: unreliable channels, resilient channels and operational channels. No assumptions have to be made about *unreliable channels*:

[1]The term *fairness* will be used in two different contexts in this paper that must not be confused: on one hand it is used to denote the fairness property that a protocol must respect, on the other hand we use it when discussing fair computations in the context of temporal logic.

data may be lost. A *resilient channel* delivers data correctly after a finite, but unknown amount of time. Data may be delayed, but will eventually arrive. When using an *operational channel* data arrive correctly before a known, constant amount of time. Operational channels are however rather unrealistic in heterogeneous networks. A formal definition of both these properties and the channels, will be found later.

In comparison to other security issues, such as privacy or authenticity of communications, non-repudiation has not been studied intensively. However, the more studied authentication protocols have shown that the design of security protocols is an error prone process. Some flaws have only been found after years (e.g. the Needham-Schroeder public-key authentication protocol [15]). Hence, a need for formal verification methods has been identified. These methods include an adequate specification, more precise than the traditional, informal and sometimes ambiguous one often used in literature. Another aim is the automatic verification. Several works have been engaged: believe logics, such as BAN [9], general purpose model checkers, e.g. FDR [16], theorem provers as ISABELLE for instance [21] and also special purpose verification tools as the NRL protocol analyzer [19] have successfully been applied to authentication protocols. To the best of our knowledge only four attempts have been made to verify non-repudiation and fair exchange protocols. First works have been done on non-repudiation protocols using CSP [22], where the proofs were done by hand, and Zhou and Gollmann briefly considered using the belief logic SVO [29]. Some work on fair exchange protocols has been realized using the model-checker Murφ [23, 24, 25] as well as the animation tool Possum [8].

Non-repudiation protocols as games. There are some fundamental differences between authentication protocols and exchange protocols, e.g. non-repudiation protocols. Generally one of the most difficult problems in authentication protocols is to deal with the presence of an intruder. In non-repudiation protocols we do not need to model an intruder, but we have to consider that either Alice or Bob, the two entities taking part in the protocol, may *cheat* (see also [22] and [8]). The most important difference is that exchange protocols, above all optimistic ones, are not linear. Generally authentication protocols are ping pong protocols and only allow very few different traces. However, optimistic non-repudiation and fair exchange protocols are divided in several subprotocols, making branching possible, although they are intended to be executed in a given order by a honest entity. Most of the recent protocols give to Alice and Bob the possibility to *abort* or *recover*, i.e. force a successful ending, a protocol. Hence, it is not always easy to foresee all possible interleavings of these protocols, or even be sure at which moment in the protocol these subprotocols can be launched. Changing the order or the moment of execution could result in subtle errors. This is the reason why we propose a new method for the specification and the verification of exchange protocols. First, we want to model the actions that are possible in the course of the protocol and not stick to a given predefined order of execution. In that way, we give a malicious entity the potential not to follow the protocol, but to construct an attack against the honest entity. Second, we consider the execution of the protocol as a game: each entity (Alice, Bob, TTP) and each communication channel is a player. Using games in the context of fair exchange protocol design has first been considered by Asokan et al. in [5]. We can think of designing a protocol as finding a *strategy*: the strategy proposed by the protocol has to defend a honest entity against all possible strategies of malicious parties that are trying to cheat. This point of view also allows us to express formally the required properties as strategies. For instance, a property such as fairness for Alice can be expressed as follows: "a coalition of Bob and all the communication channels does not have a strategy to obtain a non-repudiation of origin evidence without Alice having a strategy to obtain a non-repudiation of receipt evidence". Here, we have rephrased the property as the existence of a strategy. The main advantage of modeling such protocols as games is that we *directly* and *formally* take into account the possibility of *adversarial* behaviors. The other participating entities can try to fool Alice when she tries to obtain her non-repudiation of receipt evidence. As the communication channels are also modeled as players, they can cooperate with protocol entities. This means that either the channels are not well-functioning or that they are controlled by a player of the coalition. For instance, when a message is lost on an unreliable channel, this can be due to a network failure, or a dishonest player who removed the message by cooperating with the communication channels. As a second example, consider the game view of the viability property: Alice, in cooperation with Bob must have a strategy against the coalition of the communication channels, such that both Alice and Bob possess their expected evidences at the end of the protocol. This example illustrates the facility using the strategies not only to express adversarial behaviors, but also *cooperative* behaviors between several players. The trust in the TTP is modeled by giving the TTP a unique strategy: even if he makes part of one coalition or the other, his behavior is deterministic and thus *cannot choose* to help anyone. By using alternating transition systems and alternating-time temporal logic of Alur et al. [1], we are able to formalize the non-repudiation protocols and their requirements in a direct way.

Structure of the paper. We organize the paper as follows. In section 2, we introduce the *alternating transition systems* and the *alternating-time temporal logic* of Alur et al. [1]. The alternating transition systems are used to formalize exchange protocols as games and alternating-time temporal logic is used to formalize the requirements that exchange protocols must ensure. Section 3 shows how exchange protocols, and more specifically non-repudiation protocols, can be modeled *naturally* and *accurately* as games. Viewing those protocols as games allows us to define the requirements as existence of strategy for players to reach a given goal. The players are the different entities participating in the protocol. In that way, we *directly* and *formally* take into account the possible existence of adversarial (cheating) behaviors of participating entities to the protocol. The main properties that a non-repudiation protocol must ensure are given as alternating-time temporal logic formulas. In section 4, we report on results about the automatic verification of several non-repudiation protocols using the model-checker MOCHA [2]. MOCHA is a model-checker that supports the alternating transition systems and the alternating-time temporal logic. We use this tool to formally analyze four protocols and were able to explain in term of strategies several bugs. Then, we compare our techniques to some related works. Finally, we report on plans for future works in section 5 and draw some conclusions.

2 A Formal Model of Games and its Logic

Alternating transition systems. The formalism that we use to model exchange protocols as games is the model of alternating transition systems [1], ATS for short. ATS are a game variant of usual Kripke structures. An ATS is a 6-tuple $S = \langle \Pi, \Sigma, Q, Q_0, \pi, \delta \rangle$ with the following components:

- Π is a set of propositions.
- Σ is a set of players.
- Q is a set of states.
- $Q_0 \subseteq Q$ is a set of initial states.
- $\pi : Q \rightarrow 2^{\Pi}$ maps each state to the set of propositions that are true in the state.

- $\delta : Q \times \Sigma \to 2^{2^Q}$ is a transition function that maps a state and a player to a nonempty set of choices, where each choice is a set of possible next states. Whenever the system is in state q, each player a chooses a set $Q_a \in \delta(q, a)$. In this way, a player a ensures that the next state of the system will be in its choice Q_a. However, which state in Q_a will be next depends on the choices made by the other players, because the successor of q must lie in the intersection $\bigcap_{a\in\Sigma} Q_a$ of the choices made by all players. The transition function is non-blocking and the players together choose a unique next state. Thus, we require that this intersection always contains a unique state: assuming $\Sigma = \{a_1, \ldots, a_n\}$, then for every state $q \in Q$ and every set $Q_1, \ldots, Q_n$ of choices $Q_i \in \delta(q, a_i)$, the intersection $Q_1 \cap \ldots \cap Q_n$ is a singleton.

For two states q and q' and a player a, we say that q' is an *a-successor* of q if there exists a set $Q' \in \delta(q, a)$ such that $q' \in Q'$. We denote by $succ(q, a)$ the set of a-successors of q. For two states q and q', we say that q' is a *successor* of q if for all players $a \in \Sigma$, we have $q' \in succ(q, a)$. Thus, q' is a successor of q iff whenever the system S is in state q, the players in Σ can cooperate so that q' will be the next state. A *computation* of S is an infinite sequence $\lambda = q_0, q_1, q_2, \ldots$ of states such that for all positions $i \geq 0$, the state q_{i+1} is a successor of the state q_i. We refer to a computation starting at state q as a *q-computation*. For a computation λ and a position $i \geq 0$, we use $\lambda[i]$, $\lambda[0, i]$, and $\lambda[i, \infty]$ to denote the i-th state in λ, the finite prefix $q_0, q_1, \ldots, q_i$ of λ, and the infinite suffix $q_i, q_{i+1}, \ldots$ of λ, respectively.

Alternating-time temporal logic. The *Alternating-time Temporal Logic* [1] (ATL for short) is defined with respect to a finite set Π of *propositions* and a finite set Σ of *players*. An ATL formula is one of the following:

- p, for propositions $p \in \Pi$.
- $\neg\varphi$ or $\varphi_1 \vee \varphi_2$, where φ, φ_1, and φ_2 are ATL formulas.
- $\langle\langle A\rangle\rangle \bigcirc \varphi$, $\langle\langle A\rangle\rangle \Box\varphi$, or $\langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2$, where $A \subseteq \Sigma$ is a set of players, and φ, φ_1, and φ_2 are ATL formulas.

The operator $\langle\langle\ \rangle\rangle$ is a *path quantifier*, and $\bigcirc$ ("next"), $\Box$ ("always"), and $\mathcal{U}$ ("until") are *temporal operators*. The logic ATL is similar to the branching-time temporal logic CTL, only that path quantifiers are parameterized by sets of players. In fact, the parameterized path quantifier $\langle\langle A\rangle\rangle$ is a generalization of the path quantifiers of CTL: the existential path quantifier $\exists$ corresponds to $\langle\langle\Sigma\rangle\rangle$, and the universal path quantifier $\forall$ corresponds to $\langle\langle\emptyset\rangle\rangle$. Sometimes we write $\langle\langle a_1, \ldots, a_n\rangle\rangle$ instead of $\langle\langle\{a_1, \ldots, a_n\}\rangle\rangle$. Additional boolean connectives are defined from $\neg$ and $\vee$ in the usual manner. As in CTL, we write $\langle\langle A\rangle\rangle \Diamond\varphi$ for $\langle\langle A\rangle\rangle\, true\, \mathcal{U} \varphi$. Extensions of ATL as ATL* and the alternating μ-calculus are defined in [1].

We interpret ATL formulas over the states of a given ATS S that has the same propositions and players. The labeling of the states of S with propositions is used to evaluate the atomic formulas of ATL. The logical connectives $\neg$ and $\vee$ have the standard interpretation. To evaluate a formula of the form $\langle\langle A\rangle\rangle\psi$ at a state q of S, consider the following two-player game between a protagonist and an antagonist. The game proceeds in an infinite sequence of rounds, and after each round, the position of the game is a state of S. The initial position is q. Now consider the game in some position u. To update the position, first the protagonist chooses for every player $a \in A$, a set $Q_a \in \delta(u, a)$. Then, the antagonist chooses a successor v of u, such that $v \in Q_b \in \delta(u, b)$, for all $b \in \Sigma \setminus A$, and $v \in Q_a$ for all $a \in A$, and the position of the game is updated to v. In this way, the game continues forever and produces a computation. The protagonist wins the game if the resulting computation satisfies the subformula ψ, read as a linear temporal formula whose outermost operator is $\bigcirc$, $\Box$, or $\mathcal{U}$. The ATL formula $\langle\langle A\rangle\rangle\psi$ holds at the state q if the protagonist has a winning strategy in this game.

In order to define the semantics of ATL formally, we first define the notion of strategies. Consider an ATS $S = \langle \Pi, \Sigma, Q, \pi, \delta\rangle$. A *strategy* for a player $a \in \Sigma$ is a mapping $f_a : Q^+ \to 2^Q$ such that for all $\lambda \in Q^*$ and all $q \in Q$, we have $f_a(\lambda \cdot q) \in \delta(q, a)$. Thus, the strategy f_a maps each finite prefix $\lambda \cdot q$ of a computation to a set in $\delta(q, a)$. This set contains possible extensions of the computation as suggested to player a by the strategy. Each strategy f_a induces a set of computations that player a can enforce. Given a state q, a set A of players, and a set $F_A = \{f_a \mid a \in A\}$ of strategies, one for each player in A, we define the *outcomes* of F_A from q to be the set $out(q, F_A)$ of all q-computations that the players in A can enforce when they cooperate and follow the strategies in F_A; that is, a computation $\lambda = q_0, q_1, q_2, \dots$ is in $out(q, F_A)$ if $q_0 = q$ and for all positions $i \geq 0$, the state q_{i+1} is a successor of q_i satisfying $q_{i+1} \in \bigcap_{a\in A} f_a(\lambda[0, i])$.

We can now turn to a formal definition of the semantics of ATL. We write $S, q \models \varphi$ ("state q satisfies formula φ in the structure S") to indicate that the formula φ holds at state q of S. When S is clear from the context we omit it and write $q \models \varphi$. The relation $\models$ is defined, for all states q of S, inductively as follows:

- For $p \in \Pi$, we have $q \models p$ iff $p \in \pi(q)$.
- $q \models \neg\varphi$ iff $q \not\models \varphi$.
- $q \models \varphi_1 \vee \varphi_2$ iff $q \models \varphi_1$ or $q \models \varphi_2$.
- $q \models \langle\langle A\rangle\rangle\bigcirc\varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \models \varphi$.
- $q \models \langle\langle A\rangle\rangle\Box\varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$ and all positions $i \geq 0$, we have $\lambda[i] \models \varphi$.
- $q \models \langle\langle A\rangle\rangle\varphi_1\mathcal{U}\varphi_2$ iff there exists a set F_A of strategies, one for each player in A, such that for all computations $\lambda \in out(q, F_A)$ there exists a position $i \geq 0$ such that $\lambda[i] \models \varphi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \models \varphi_1$.

Note that the next operator $\bigcirc$ is local: $q \models \langle\langle A\rangle\rangle\bigcirc\varphi$ iff for each player $a \in A$ there exists a set $Q_a \in \delta(q, a)$ such that for each state $q' \in \bigcap_{a\in A} Q_a$, if q' is a successor of q, then $q' \models \varphi$.

We say that the ATS S satisfies φ, noted $S \models \varphi$, iff $\forall q \in Q_0 : q \models \varphi$.

We now illustrate the expressive power of ATL. Consider the set of players $\Sigma = \{a, b, c\}$ and the following formulas with their verbal reading:

- $\langle\langle a\rangle\rangle\Diamond p$, player a has a strategy against players b and c to eventually reach a state where the proposition p is true;
- $\neg\langle\langle b, c\rangle\rangle\Box p$, the coalition of players b and c does not have a strategy against a to maintain the proposition p true for ever;
- $\langle\langle a, b\rangle\rangle\bigcirc(p \wedge \neg\langle\langle c\rangle\rangle\Box p)$, a and b can cooperate so that the next state satisfies p and from there c does not have a strategy to impose p for ever.

Those three formulas are a good illustration of the great expressive power of ATL to express cooperative as well as adversarial behaviors between players.

Fairness. As in the usual temporal logic setting, fairness can be used to rule out computations. For example, to evaluate a formula of the form $\langle\langle a\rangle\rangle\Diamond\varphi$, we may want to only consider computations where the antagonists, that is the agents in $\Sigma \setminus \{a\}$, respect their fairness constraints. Consider a computation where a resilient channel has continuously the choice to deliver a message but never does so. A *fairness constraint* is a function $\gamma : Q \times \Sigma \to 2^{2^Q}$ such that for each $q \in Q$ and $a \in \Sigma$, we have $\gamma(q, a) \subseteq \delta(q, a)$. Given a computation $\lambda = q_0q_1 \ldots q_n \ldots$, we say that γ is *a-enabled* at position $i \geq 0$ if $\gamma(q_i, a) \neq \emptyset$. We say that γ is *a-taken* at position $i \geq 0$ if there exists a set $Q' \in \gamma(q_i, a)$ such that $q_{i+1} \in Q'$. Finally given a set $A \subseteq \Sigma$, we say that λ is *weakly* $\langle\gamma, A\rangle$*-fair* [2] if for each agent $a \in A$, either there are infinitely many positions of λ at which γ is not a-enabled, or there are infinitely many positions of λ at which γ is a-taken. A *fairness condition* Γ is a set of fairness constraints. So, for a given set $A \subseteq \Sigma$, we say that λ is *weakly* $\langle\Gamma, A\rangle$*-fair*, if λ is weakly $\langle\gamma, A\rangle$-fair for all fairness constraints $\gamma \in \Gamma$. Given an ATS S, and a (weak) fairness condition Γ, we define the semantics of Fair ATL as follows:

- $q \models_F \langle\langle A\rangle\rangle\bigcirc\,\varphi$ iff there exists a set F_A of strategies, one for each agent in A, such that for all $\langle\Gamma, \Sigma \setminus A\rangle$-fair computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \models_F \varphi$.
- $q \models_F \langle\langle A\rangle\rangle\Box\varphi$ iff there exists a set F_A of strategies, one for each player in A, such that for all $\langle\Gamma, \Sigma \setminus A\rangle$-fair computations $\lambda \in out(q, F_A)$ and all positions $i \geq 0$, we have $\lambda[i] \models_F \varphi$.
- $q \models_F \langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\varphi_2$ iff there exists a set F_A of strategies, one for each player in A, such that for all $\langle\Gamma, \Sigma \setminus A\rangle$-fair computations $\lambda \in out(q, F_A)$ there exists a position $i \geq 0$ such that $\lambda[i] \models_F \varphi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \models_F \varphi_1$.
- the semantics of the other operators is as in standard ATL.

In [1], a model-checking algorithm handling fairness constraints is presented.

Game guarded command language. Modeling a protocol directly with an alternating transition system would be cumbersome. Instead we use a simple modeling formalism based on Dijkstra's guarded command language, previously introduced in [12]. This simple modeling language is a subset of the reactive modules used by the verification tool Mocha. Let X be a set of variables interpreted over some possibly infinite domains. We denote by $X' = \{x' \mid x \in X\}$ the set of variables obtained by priming each variable in X. Those variables are used to refer to the value of the variables in the next state. A valuation v of the variables X is a function which maps each of the variables in X into a value in its domain. Denote by $\mathcal{V}_X$ the set of all possible valuations of the variables X. For any valuation v, we write v' the valuation obtained by priming each domain variable of v. For any predicate φ over the variables X and a valuation $v \in \mathcal{V}_X$, denote by $\varphi[\![v]\!]$ the truth value of the predicate with all the free variables of the program P interpreted according to the valuation v. For $Y \subseteq X$, we define $v_{|Y}$ as the valuation with $dom(v_{|Y}) = Y$, such that $\forall y \in Y : v_{|Y}(y) = v(y)$.

Given a set of program variables X, a Y*-action* ξ (guarded command) has the form $[\!]$ *guard* $\rightarrow$ *update*, where $Y \subseteq X$, the guard *guard* is a boolean predicate over X, and the update relation *update* is a boolean predicate over $X \cup Y'$. We also write $guard_\xi$ and $update_\xi$ to represent the guard and update relation of ξ respectively. Given a valuation $v \in \mathcal{V}_X$, the action ξ is said to be *enabled* if the guard $guard_\xi[\![v]\!]$ evaluates to true. We assume that for every Y-action, the update relation is functional, that is, for all valuation functions $v \in \mathcal{V}_X$, there is exactly one valuation function

[2] Note that there exists also, as in the usual temporal logic setting a notion of strong fairness, the interested reader is referred to [1] for the definition. We will not need the notion of strong fairness in this paper.

$u \in \mathcal{V}_Y$ such that $update_\xi[\![v \cup u']\!]$ holds. A *Y-process* Φ is a finite set of Y-actions. We say that the set of variables Y is *the controlled variables* of Φ. We require that for any valuation of the program variables, at least one action in the process is enabled.

To model multi-player games, we partition the program variables X into disjoint sets $X_1, X_2, \ldots, X_n$, i.e., $X = X_1 \uplus X_2 \uplus \cdots \uplus X_n$. Intuitively, X_i contains the variables updated by player i. A *program* $P = (\Phi_1, \Phi_2, \ldots, \Phi_n)$ over X is a tuple of processes such that Φ_i is an X_i-process. Each process of the program can be defined by composing smaller processes, each of which controls a subset of the variables. Moreover we define a boolean predicate *init*, that defines the initial values of the variables.

The *semantics* of a program $P = \{\Phi_1, \Phi_2, \ldots \Phi_n\}$ over the program variables X is the alternating transition system S_P:

- $\Sigma = \{1, 2, \ldots, n\}$ (the n players of the program P).
- $Q = \mathcal{V}_X$, i.e., a state of S_P is a valuation of the program variables.
- $Q_0 = \{v \in \mathcal{V}_X | init[\![v]\!]\}$.
- $\delta : \mathcal{V}_X \times \Sigma \rightarrow 2^{2^{\mathcal{V}_X}}$ is the transition function defined as $\delta(v, i) = \big\{\{r \mid r_{|X_i} = w \wedge r \in \mathcal{V}_X\} \mid \xi \in \Phi_i \wedge guard_\xi[\![v]\!] \wedge update_\xi[\![v \cup w']\!]\big\}$.
- Π is a set of boolean predicates over the program variables X.
- π maps a state q, which is a valuation, to the set $\pi(q)$ of propositions that the valuation satisfies.

So, for $i \in \Sigma = \{1, 2, \ldots, n\}$, player i controls the actions in Φ_i.

A program is run in steps. At each step, player 1 chooses an enabled action $\xi \in \Phi_1$, and updates the values of the variables in X_1 according to the predicate $update_\xi$. Independently and simultaneously, player 2 chooses an enabled action $\eta \in \Phi_2$ and updates the values of the variables in X_2 according to $update_\eta$. And so on for the $n - 2$ other players.

We introduce fairness in our modeling language at the level of actions. Given a set of actions Φ_i of player i, a fairness constraint for player i is a subset $\Gamma_i \subseteq \Phi_i$ of actions in regard of which, player i must be (weakly) fair. Intuitively, an action that belongs to Γ_i can not be constantly enabled without being taken by player i. So, a fairness condition for a program $P = \{\Phi_1, \Phi_2, \ldots, \Phi_n\}$ is a set $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$ where $\Gamma_i \subseteq \Phi_i$ for $i : 1 \leq i \leq n$. This set of subsets of actions of the program P induces the fairness condition Γ on the ATS defining the semantics of P whose transition relation is δ. Γ contains a fairness constraint γ_ξ, for each action $\xi \in \Gamma_i$, defined as follows:

$$\gamma_\xi(v, j) = \begin{cases} \emptyset & \text{if } i \neq j \\ \emptyset & \text{if } i = j \text{ and } \neg guard_\xi[\![v]\!] \\ \{Q' \mid Q' \in \delta(v, j) \wedge \forall u' \in Q' : update_\xi[\![v \cup u']\!]\} & \text{if } i = j \text{ and } guard_\xi[\![v]\!]. \end{cases}$$

Illustration. In the example of figure 1, we use the following syntactic assymptions. First, in the update part of the command, if a variable x does not appear in the right-hand side of the command, it is considered as unchanged by the command, so implicitely, we have $x = x'$ for each such variable. Second, the command *true* $\rightarrow$ means that the player owning this command can choose to leave unchanged its controlled variables, this is a kind of "idle" action. Note that as the guard is true, that action can be taken at any step of the game if no fairness constraints are imposed on the player.

```
A:
[] true → p':=true
[] true →

B:
[] p → q':=true
[] true →

C:
[] q → r':=true
[] true →
```

Figure 1: A simple program in the game guarded command language.

We are now equipped to consider the example. The variable p is controlled by player A, q by B and r by C. A state of the system, is a valuation for the propositions p, q, r. Let us consider the state v such that $v_p = \mathit{false}$, $v_q = \mathit{false}$, and $v_r = \mathit{false}$. From that state, players A, B, and C can cooperate so that the system reach a state where r is true. Formally, this is expressed by the following statement:

$$v \models \langle\!\langle A, B, C \rangle\!\rangle \Diamond r \tag{1}$$

But as player C can always choose to perform the "idle" action, then A and B alone do not have a strategy to reach a state where r is true, so

$$v \models \neg \langle\!\langle A, B \rangle\!\rangle \Diamond r \tag{2}$$

In fact, player C has a spoiling strategy to prevent A and B to reach a r-state. Furthermore, he has a winning strategy to impose $\neg r$ for ever[3]:

$$v \models \langle\!\langle C \rangle\!\rangle \Box \neg r \tag{3}$$

The winning strategy simply consists in taking the idle action at every step of the game. If we want to exclude this behavior (because it is not a realistic one for example), we can use fairness constraints. By setting a weak fairness constraint on the action of C that set r to true, we exclude the strategies of C that consists in taking the idle action for ever. This fairness constraint excludes spoiling strategies of C for formula 2. So, we have :

$$v \models_F \langle\!\langle A, B \rangle\!\rangle \Diamond r \tag{4}$$

3 Formal Modelization of Non-Repudiation Protocols

Modeling of the protocols. Non-repudiation and fair exchange protocols have several particularities that allow us to make some simplifications when analyzing them. As also noted in [8] all

[3]The carefull reader has noted that statement $v \models \neg \langle\!\langle A, B \rangle\!\rangle \Diamond r$ is not equivalent to $\langle\!\langle C \rangle\!\rangle \Box \neg r$. We have that $\langle\!\langle C \rangle\!\rangle \Box \neg r$ implies $v \models \neg \langle\!\langle A, B \rangle\!\rangle \Diamond r$ but not the other way round.

messages appearing in these protocols are generally protected by digital signatures or an equivalent mechanism. Hence we consider that only well formed messages can be sent. We suppose that messages, having no valid signatures, or that are not coherent with previous messages will be ignored and are not modelled at all. Moreover each protocol execution can be uniquely identified by the identity of the participating entities and a special label. When the label contains some well chosen information related to an execution, as proposed for instance in [30], it is possible to show that different parallel executions can not interfere. A unique identification is inherent to optimistic protocols, as the TTP must be able to identify the protocol run, in order not to mix abort or recovery requests from different protocol runs. Therefore we do not consider verification of several parallel runs.

The notation, classically used in literature to describe cryptographic protocols ($A \rightarrow B : m$, to denote that Alice sends a message m to Bob), has several drawbacks. The protocols are presented as a linear sequence of message exchanges, with a predefined order. In the case of optimistic exchange protocols often subprotocols can be invoked at different moments. Unfortunately, running a subprotocol at a time not foreseen by the designer, may have unexpected side-effects, threatening the security of a protocol if one of the participating entities tries to cheat the other entities. Therefore, we use the modeling language described in the previous section to model the exchange protocols. Remember that to execute a given action ξ, $guard_\xi$ must evaluate to true. The guard $guard_\xi$ is used to represent the elements (such as keys, messages, ...) necessary for a participating entity, that is a player in our modelization, to execute the action ξ. The variables controlled by a player represent his current knowledge and thus determine which actions he/she can execute. This specification takes into account all possible executions of subprotocols, with respect to the power we give to a player and his initial knowledge, as we do not give any predefined order to these guarded commands. At each point in the protocol execution all messages that could possibly be sent are determined. Thus, the specification enables a malicious entity to choose a different order of execution and construct a possible attack. We model the two players Alice and Bob with this approach.

The TTP is a special player and has to be modeled in a particular way. The TTP must be *impartial*, it may not help one or the other player. To make sure that the TTP does not have a strategy to help one of the players to cheat, we model the TTP in a deterministic way: at each stage of the execution of the protocol, the TTP executes the action requested by the protocol. This can easily be done. As an example, consider a protocol where the TTP has to execute the actions $\xi_1, \xi_2, ...\xi_n$ in this given order. We only have to use a n valued variable initialized to 1; each time that the TTP takes an action this counter is incremented and the guards of the actions of the TTP ensure that only this sequence is possible.

The communication channels can also be modeled as players. Each transmission is modeled as a guarded command. We consider three different kinds of communication channels: unreliable, resilient, and operational. To specify *unreliable channels* we add an action that the channel can always take and has no effect, i.e. this is simply an idle action. As a consequence, a message sent on an unreliable channel may never be delivered. Hence, a malicious player controlling an unreliable channel can prevent messages from arriving. A *resilient channel* can be specified in a similar way, but we have to add fairness constraints on the computations in order to force the transmission before a finite amount of time. More precisely, assume that a resilient channel c has to deliver a message m_1 when the proposition $\mathtt{SendM_1}$ is set to true and the delivery of the message is modeled by setting $\mathtt{M_1}$ to true. Hence, if Γ denotes the fairness condition of the program P, modeling the protocol, we have that $[\![\ \mathtt{SendM1} \rightarrow \mathtt{M1} \in \Gamma_c$ and $\Gamma_c \in \Gamma$. This fairness constraint rules out trajectories where c can deliver a message and never delivers it. On resilient channels, it is not possible to alter data or to loose a message. Messages can be delayed, but arrive before a finite amount of time. *Operational channels* do not have any additional action and thus immediately transmit the messages that have been sent on it. By immediately transmitting messages we alter the channel definition, which requires messages to be transmitted before a known constant time. This is however acceptable as the sender can always wait before sending the message to obtain the desired delay. As on resilient channels, data cannot be altered or lost. Moreover, messages are immediately transmitted. Note that none of these channels provide confidentiality: it is possible to eavesdrop them. Confidentiality could however easily added by either altering the protocol messages, adding encryption, or by restricting a malicious entity to eavesdrop a given channel.

When modelling Alice and Bob, we model two versions of each of them: a honest one and a malicious one. When starting to model a malicious player from an informally described protocol we apply the following method. First, we determine the initial knowledge of each player. Initial knowledge may for instance include nonces, public and shared keys as well as initially known messages. All these items are represented by boolean variables initialized to true. All other variables are set to false, indicating that there value is not known yet. Then we model all *useful* cryptographic operations that can be applied on the initial knowledge. Note that we assume perfect cryptography. Cryptographic primitives are treated as black boxes and cryptanalysis is not considered in our model. Moreover, the term *useful* means that the operation generates an item that can be used later in the protocol. This means that the behaviour of the entities is restricted in comparison to a full Dolev-Yao intruder. We restrict the capabilities of the adversary to the operations, that permit to construct messages, that can be *accepted* by an other entity. As in this kind of protocols generally all messages are signed, the set of messages, we consider as *useful* is rather restricted. We conjecture that this restricted set of messages is sufficient. Now the following steps are applied to each message $\mathtt{X} \rightarrow \mathtt{Y} : \mathtt{M_i} = \mathtt{m_{i1}}, \mathtt{m_{i2}}, \ldots, \mathtt{m_{in}}$ of the protocol. To represent the sending of $\mathtt{M_i}$, a guarded command $\mathtt{m_{i1}} \wedge \cdots \wedge \mathtt{m_{in}} \rightarrow \mathtt{SENDm'_i} := \mathtt{true}$ is added to player $\mathtt{X}$'s description. The guard contains the conjunction of all the elements of the message. This means that a player is able to send a message if and only if he has knowledge of all the required elements. The update relation expresses the intention to send the message. The transmission of the message is represented by a guarded command added to the description of the communication channel between $\mathtt{X}$ and $\mathtt{Y}$. The command will be of the form $\mathtt{SENDm_i} \rightarrow \mathtt{M'_i} := \mathtt{true}$. Note that in order to delay this message a communication channel may choose to execute a different command, typically $\mathtt{true} \rightarrow$. Reception of the message is modeled by adding the following command to $\mathtt{Y}$'s description $\mathtt{M_i} \rightarrow \mathtt{m_{i1}} := \mathtt{true}; \ldots; \mathtt{m_{in}} := \mathtt{true}$. For each element of $\mathtt{M_i}$ being set to true for the first time in $\mathtt{Y}$'s description, we also add all useful cryptographic operations that can be applied on it. Remember that malicious entities have the ability to eavesdrop channels. A malicious entity can read all messages, even if they are not destined to him. Of course, this does not mean that he can decipher an encrypted message if he does not possess the decryption key. It merely means that he can read the ciphertext. Eavesdropping is achieved by adding guarded commands, that allow him to receive messages sent to other entities, e.g. to the TTP. Hence, a malicious entity is able to send messages out of order, schedule resilient communication channels, prevent arrival on unreliable channels and eavesdrop channels.

As an illustration, let us consider the guarded commands of figure 2. The first guarded command models the encryption of message $\mathtt{Ma}$ under key $\mathtt{Ka}$ resulting in the cipher $\mathtt{Ca}$. The second one expresses that "if Alice has knowledge of $\mathtt{L}$, $\mathtt{T}$, $\mathtt{C}$ and $\mathtt{EOO}$, she can send message 1". The third guarded command models that the communication channel deliver the message 1 ($\mathtt{M1}$ becomes true). The last guarded command expresses that Bob increases his knowledge when he receives the message $\mathtt{M1}$. The meaning of the variables will become clearer in the next section.

The method above describes how to model Alice, respectively Bob having an arbitrary behavior,

$\texttt{Alice}:$
$\vdots$
$[]\ \texttt{Ma} \wedge \texttt{Ka} \rightarrow \texttt{Ca}' := \texttt{true}$
$\vdots$
$[]\ \texttt{La} \wedge \texttt{Ta} \wedge \texttt{Ca} \wedge \texttt{EOOa} \wedge \neg\texttt{STOPa} \wedge \neg\texttt{SENDm1} \rightarrow \texttt{SENDm1}' := \texttt{true};$
$\vdots$
$\texttt{Communication Channel}:$
$\vdots$
$[]\ \texttt{SENDm1} \wedge \neg\texttt{M1} \rightarrow \texttt{M1}' := \texttt{true};$
$\vdots$
$\texttt{Bob}:$
$\vdots$
$[]\ \texttt{M1} \wedge \neg\texttt{EOOb} \wedge \neg\texttt{STOPb} \rightarrow \texttt{Lb}' := \texttt{true}; \texttt{Tb}' := \texttt{true}; \texttt{Cb}' := \texttt{true}; \texttt{EOOb}' := \texttt{true};$
$\vdots$

Figure 2: Example of guarded commands used for protocol modeling.

with respect to the abilities of a malicious participant. To restrict their behavior to the honest protocol execution, we can easily reinforce the guards so that the order of execution corresponds to the one dictated by the protocol and remove the commands modeling eavesdropping. We achieve two different descriptions of Alice, as well as of Bob: the first description models arbitrary, possibly malicious, behavior, the second one only the honest behavior.

Modeling of requirements. We show here how the main requirements that an exchange protocol must fulfill, can be *naturally* rephrased as the existence of strategies for the participating entities to reach their goal. The logic ATL is used to formalize those requirements. We will concentrate here on properties of non-repudiation protocols. The properties described are general properties that do not apply to a given protocol. They may need to be instantiated when studying a particular protocol.

In a first approximation, we introduced fairness as the property that either both entities receive all their desired evidences or none of them receives any valuable evidence. We can now formulate this requirement as the existence of strategies. We say that the protocol is fair for Alice if "Bob and the communication channels do not have a strategy to reach a state where Bob has his proof of origin and Alice has no more a strategy to obtain her proof of receipt". This can be formally expressed by the following ATL formula:

$$\neg\langle\langle B, \mathsf{Com}\rangle\rangle\Diamond(\mathsf{NRO} \wedge \neg\langle\langle A\rangle\rangle\Diamond\mathsf{NRR}) \tag{5}$$

Here, Com denotes the set of all communication channels. NRR and NRO respectively denote the non-repudiation of receipt and origin evidences. This is a generic notation as the form of those evidences depends on a particular protocol. The expressive power of the logic ATL is well illustrated by this example. Cooperation and adversarity are naturally expressed. Firstly, we let both Alice and Bob have arbitrary behavior. If fairness is not verified, this means that the protocol contains a flaw. On the other hand, if the formula holds, this only means that Alice has a strategy to defend herself, against a malicious entity, but it does not mean that this strategy is the strategy dictated by the protocol. Therefore, we should replace Alice by the honest version of her behaviour in the above given formula. Note that even if Alice is modelled in a honest way, this generally leaves certain non-deterministic choices, due to the common "underspecification" of the protocols. For instance, it is not always clearly defined, where the protocol can be stopped, what values should be used for the time-outs, etc. So, verifying the formula with honest Alice, only verifies whether Alice can resolve her internal choices, that all belong to the honest behavior, in order to achieve fairness. When this formula is not verified, a flaw is found. However, when it holds, this only reflects the fact that Alice is able to *defend* herself against Bob. To show that each implementation is correct we have to verify stronger formulas. Before discussing these stronger formulas let us consider the following CTL formula, in order to better understand the advantages of using a game logic to formalize this requirement:

$$\neg\exists\Diamond(\mathsf{NRO} \wedge \neg\exists\Diamond\mathsf{NRR}) \tag{6}$$

This may look as an appropriate CTL candidate for the formalization of fairness for Bob. Note that we have obtained this formula by replacing the two teams in (5) by the entire set of players Σ. Note also that the formula (6) can be rewritten in the equivalent and more readable positive form:

$$\forall\Box(\mathsf{NRO} \rightarrow \exists\Diamond\mathsf{NRR}) \tag{7}$$

That says "on every state of every run of the protocol, if Bob has his evidence of origin then there should exist a continuation of the protocol on which Alice eventually receives her evidence of receipt". This way, we have lost however the ability of distinguishing between cooperative and adversarial behaviors. Let us show what are some consequences of this, so to say, "loss of precision". For example, it may be the case that the formula (7) is verified because in the course of the protocol a *resilient* channel helps Alice to obtain her proof by delivering a message within a given bound (which is not generally the case for such a channel). As this assumption of cooperative behavior of the resilient channels can not be made in general, in order to be fair the protocol should guarantee to Alice that she obtains her proof even if the resilient channel does not deliver the message within a certain bound. This fact is not ensured by the CTL formalization but is ensured by our ATL formalization. In fact, in formula (5), it is explicitly stated that the communication channels play against Alice. A resilient channel has only the obligation to deliver the message after a finite amount of time and not within a given bound, even if that helps Alice to obtain her proof. This non-cooperative aspect is precisely formalized in ATL. Also formula (6) may be verified because there exists some paths, where Bob is honest and allows Alice to obtain her proof. Again, the protocol should not make the hypothesis that Bob is honest, and should ensure fairness for Alice even if Bob is trying to cheat her. This adversarial behavior is formally modeled by our ATL formula and cannot be formalized in CTL. We can also have that the CTL formula is false, even if the ATL formula holds. Such a situation can occur if Alice's specification contains some non-determinism. For instance, if the specification does not contain at which moment Alice can stop the protocol, she may stop it at a "wrong" moment. The CTL formula will fail, while the ATL formula may hold, as not stopping the protocol at a given moment would have lead to a winning strategy. Using the notion of strategy, we "automatically" exclude behaviors of Alice that do not serve her objective. Note that this last comment also rules out the following CTL formula as a candidate for fairness:

$$\forall\Box(\mathsf{NRO} \rightarrow \forall\Diamond\mathsf{NRR}) \tag{8}$$

Situations where the protocol is not well defined, occur frequently when integrating the use of formal methods as an aid in the protocol designing process. During the first steps while designing the protocol, we generally do not want to deal with all details: the aim of the verification is to determine whether the given specification contains a winning strategy that leads to a correct protocol. This means that there exists a correct implementation, but it does not yet prove that all implementations are correct. For a more precise analysis, we have to set all, or at least most non-deterministic choices. Once this is done, a stronger formula can be checked in order to verify that *all* possible implementations of this further refined implementation are fair. One way of doing this, is to let Alice cooperate with Bob and the communication channels. This means that Alice, tries to fool herself, while respecting her restricted behavior. This results in the following formula :

$$\neg\langle\langle A_h, B, \mathsf{Com}\rangle\rangle\Diamond(\mathsf{NRO} \wedge \neg\langle\langle\emptyset\rangle\rangle\Diamond\mathsf{NRR}) \tag{9}$$

This formula is equivalent to the CTL formula (8). It means that at the end of the specification and design level, a last check should be done to assure that all possible remaining non-deterministic choices, e.g. whatever time-out is chosen, the protocol provides fairness. However, we believe that at a specification level, as well as during the design process, the fairness requirement should explicitly express the adversarial as well as the cooperative behaviors.

To complete the fairness requirement, we express fairness for Bob as follows:

$$\neg\langle\langle A, \mathsf{Com}\rangle\rangle\Diamond(\mathsf{NRR} \wedge \neg\langle\langle B\rangle\rangle\Diamond\mathsf{NRO}) \tag{10}$$

and take the conjunction of (5) and (10). We are also interested in first verifying a basic (weaker) notion of fairness: "Alice and Bob cooperate to achieve fairness". This would be translated by the following ATL formula:

$$\langle\langle A, B\rangle\rangle\Diamond((\mathsf{NRO} \wedge \mathsf{NRR}) \vee (\neg\mathsf{NRO} \wedge \neg\mathsf{NRR})) \tag{11}$$

Viability, which can be seen as a variation of the property above, is expressed by the following formula:

$$\langle\langle A, B\rangle\rangle\Diamond(\mathsf{NRO} \wedge \mathsf{NRR}) \tag{12}$$

that says "Alice and Bob can cooperate, so they are honest (i.e. they are following the protocol), and in that case the protocol should allow them, even in presence of non-cooperating channels, to obtain their respective evidences". And finally timeliness is formalized by

$$\langle\langle A\rangle\rangle\Diamond(\mathsf{stop}_A \wedge (\neg\mathsf{NRR} \rightarrow \neg\langle\langle B\rangle\rangle\Diamond\mathsf{NRO})) \tag{13}$$

which expresses that "Alice has a strategy to finish the protocol and if she does not have her evidence at that point, Bob will not be able to obtain his evidence neither". To represent the fact that Alice finishes the protocol, we introduce a special variable stop. Once stop is set to true, all of Alice's actions are disabled. In this way, we model the fact that Alice stops this protocol session, including all related sub-protocols that could be waiting for an additional message (e.g. a recovery or an abort message from the TTP) to arrive. The same requirement can be expressed for Bob.

4 Verification with Mocha

We used the model-checker Mocha to verify several protocols. As fairness constraints were not foreseen in the tool, we extended it by rewriting some of the model-checking algorithms, using the Mocha script language. We report in detail on our verification of the Zhou-Gollmann optimistic protocol [30] and illustrate on this example our verification methodology. We also checked the Asokan-Shoup-Waidner certified mail protocol [4], the Kremer-Markowitch non-repudiation protocol [13], as well as the Markowitch-Kremer multi-party non-repudiation protocol [17]. For these protocols, we focus on the results of our verification, rather than the verification process.

4.1 The ZG optimistic non-repudiation protocol

We first give an informal description of the protocol using the following notation:

- $X \to Y$: transmission from entity X to entity Y
- $X \leftrightarrow Y$: ftp get operation performed by X at Y
- $h()$: a collision resistant one-way hash function
- $E_k()$: a symmetric-key encryption function under key k
- $D_k()$: a symmetric-key decryption function under key k
- $S_X()$: the signature function of entity X
- m: the message sent from A to B
- k: the message key A uses to cipher m
- $c = E_k(m)$: the cipher of m under the key k
- $l = h(m, k)$: a label that in conjunction with the entities (A,B) identifies a protocol run
- f: a flag indicating the purpose of a message
- t: the time-out chosen by A

The protocol generates the following evidences:

- $\mathsf{EOO} = S_A(f_{\mathsf{EOO}}, B, l, t, c)$: the evidence of origin of c
- $\mathsf{EOR} = S_B(f_{\mathsf{EOR}}, A, l, t, c)$: the evidence of receipt of c
- $\mathsf{EOO}_k = S_A(f_{\mathsf{EOO}_k}, B, l, t, k)$: the evidence of origin of k
- $\mathsf{EOR}_k = S_B(f_{\mathsf{EOR}_k}, A, l, t, k)$: the evidence of receipt of k
- $\mathsf{Sub}_k = S_A(f_{\mathsf{Sub}_k}, B, l, k)$: the evidence of submission of k
- $\mathsf{Con}_k = S_{\mathsf{TTP}}(f_{\mathsf{Con}_k}, A, B, l, t, k)$: the evidence of confirmation of k issued by the TTP

The protocol is divided into two subprotocols: a main protocol and a recovery protocol. As the protocol is based on the optimistic approach, the trusted third party (TTP) does only intervene in the recovery protocol. We start by describing the main protocol, depicted in protocol 1.

Protocol 1 Main protocol

---

1. A $\to$ B: $f_{\mathsf{EOO}}, B, l, t, c, \mathsf{EOO}$
2. B $\to$ A: $f_{\mathsf{EOR}}, A, l, \mathsf{EOR}$
3. A $\to$ B: $f_{\mathsf{EOO}_k}, B, l, k, \mathsf{EOO}_k$
4. B $\to$ A: $f_{\mathsf{EOR}_k}, A, l, \mathsf{EOR}_k$

First Alice sends the digitally signed cipher $c = E_k(m)$ to Bob. In the second message Bob responds with the evidence of receipt for this cipher (EOR). If Alice does not receive the second transmission she stops the protocol, otherwise she sends the signed decryption key k to Bob. Bob answers by sending the receipt EOR_k for the key. The label l, present in each transmission, identifies the protocol run. The time-out t specified in message 1 is used in the recovery protocol. Alice may initiate the recovery protocol if Bob does not send the receipt for the key.

The steps of the recovery protocol are described in protocol 2.

Protocol 2 Recovery protocol

1. $\mathsf{A} \rightarrow \mathsf{TTP}$: $f_{\mathsf{Sub}_k}, B, l, t, k, \mathsf{Sub}_k$
2. $\mathsf{B} \leftrightarrow \mathsf{TTP}$: $f_{\mathsf{Con}_k}, A, B, l, t, k, \mathsf{Con}_k$
3. $\mathsf{A} \leftrightarrow \mathsf{TTP}$: $f_{\mathsf{Con}_k}, A, B, l, t, k, \mathsf{Con}_k$

Alice sends the signed key, together with the deadline t to the TTP. If the key arrives after t, the TTP does not accept the recovery. Otherwise the TTP publishes the key together with a confirmation Con_k for the key in a read-only accessible directory, where both Alice and Bob can fetch the key as well as Con_k. Con_k serves to Bob as the evidence of origin of the key, and to Alice as the evidence of receipt of the key as it is accessible to Bob. The deadline t is necessary for Bob to know the moment when either the key is published or will not be published anymore. In this protocol, the non-repudiation of origin evidence NRO is composed of EOO and EOO_k or of EOO and Con_k. The non-repudiation of receipt evidence NRR is composed of EOR and EOR_k or of EOR and Con_k.

Zhou et al. suppose that the channels between Alice and Bob are unreliable and the channels between the TTP and both Alice and Bob are resilient. In our model we do not model the ftp get operation, but assume that the TTP takes the initiative to send messages 2 and 3 of the recovery protocol to Alice respectively Bob, as soon as these actions are possible.

To verify the protocol we instantiate the properties defined in section 3 for two versions of both Alice and Bob. The first version allows arbitrary behavior[4], the second one restricts the behavior to the honest protocol execution. The specification of Alice, respectively Bob, allowing arbitrary behavior will be denoted with A, respectively B, while the specification of the honest behavior of Alice and Bob will be denoted $\mathsf{A_h}$, respectively $\mathsf{B_h}$. When we want to check a property, for instance fairness for Alice, we first verify that Alice has a strategy to assure the fairness requirements, when she is allowed to behave in an arbitrary way. We check that

$$\neg\langle\langle \mathsf{B}, \mathsf{Com}\rangle\rangle\Diamond(\mathsf{EOO} \wedge (\mathsf{EOO_k} \vee \mathsf{Con_k}) \wedge \neg\langle\langle \mathsf{A}\rangle\rangle\Diamond(\mathsf{EOR} \wedge (\mathsf{EOR_k} \vee \mathsf{Con_k})))$$

holds. In fact, we have to verify this formula in all states that can be reached during a protocol run where Alice behaves correctly following the protocol, e.g. after the first message of the main protocol has been sent, after the first and the second message have been sent, etc. Therefore we include into the formula until which state the protocol has been executed, without however restricting the behavior of Bob or any future behavior of Alice. Otherwise, if we only check the formula in the initial state, Alice could always decide to immediately stop the protocol, not sending any message, in order to avoid to be cheated. For instance the flaw described hereunder is found

[4]We call arbitrary behavior, all behavior that is *interesting* with respect to the protocol, i.e. behavior that results in sending or obtaining acceptable messages related to the protocol.

while verifying the following formula.

$$\neg\exists\Diamond(\mathsf{m1} \wedge \mathsf{m2} \wedge \mathsf{m3} \wedge \neg\mathsf{m4} \wedge \neg\mathsf{TimeOut} \wedge \neg\mathsf{stop_A} \wedge \neg\mathsf{stop_B} \wedge$$

$$\langle\langle \mathsf{B}, \mathsf{Com} \rangle\rangle\Diamond(\mathsf{EOO} \wedge (\mathsf{EOO_k} \vee \mathsf{Con_k}) \wedge \neg\langle\langle \mathsf{A} \rangle\rangle\Diamond(\mathsf{EOR} \wedge (\mathsf{EOR_k} \vee \mathsf{Con_k}))))$$

In the remaining of the paper, for the sake of readability, we omit to give all details in the formulas, although the formulas are slightly more complex. If the formula given above does not hold, even with arbitrary behavior of Alice, the protocol is flawed and cannot be fixed without introducing new 'mechanisms' or constraints in the protocol. If the formula holds we additionally need to check that Alice cannot be cheated when following the protocol. It is possible that the formula holds because Alice adapted a strategy, that is not the one proposed by the protocol. Therefore we check the following formula:

$$\neg\langle\langle \mathsf{B}, \mathsf{Com} \rangle\rangle\Diamond(\mathsf{EOO} \wedge (\mathsf{EOO_k} \vee \mathsf{Con_k}) \wedge \neg\langle\langle \mathsf{A_h} \rangle\rangle\Diamond(\mathsf{EOR} \wedge (\mathsf{EOR_k} \vee \mathsf{Con_k})))$$

In this formula we restricted Alice's behavior to the actions that are dictated by the protocol. If the formula evaluates to false, it means that the protocol, as originally described, is flawed. However, it is possible to easily correct it, as a strategy was found by the model-checking algorithm while checking the previous formula. This means that we do not have to fundamentally change the protocol, by adding or changing messages. A last formula we have to check is whether all possible honest behaviours are fair. Therefore we verify the following CTL formula:

$$\forall\Box((\mathsf{EOO} \wedge (\mathsf{EOO}_k \vee \mathsf{Con}_k)) \rightarrow \forall\Diamond(\mathsf{EOR} \wedge (\mathsf{EOR}_k \vee \mathsf{Con}_k)))$$

While investigating the protocol, we succeeded in finding a flaw in the protocol, even if Alice's behavior is not restricted to the honest protocol execution. The hereunder described flaw has briefly been discussed in [13]. Remember that the communication channels between the TTP and both Alice and Bob are resilient and that a finite time-out must be chosen by Alice at the begin of the protocol. In our model, the time-out is a boolean variable controlled by a player *Clock*. We restrict the time-out to become true after a finite amount of time, using a similar technique as used for transmission on resilient channels. In fact, when the main protocol has been executed until step 3, Bob has received all of his evidences and may decide to stop the protocol in order to try to cheat Alice. At this point we check whether Alice in cooperation with the clock does have a strategy against Bob who is cooperating with the communication channels, to receive her non-repudiation of receipt evidences. As Alice cooperates with the clock, she entirely controls the triggering of the time-out. Intuitively, it means that Alice can choose the time-out to occur as late as she wants, which represents the fact, that she chooses the time-out at the first step of the protocol. As Alice does not receive her evidence, she has to initiate a recovery. For the recovery protocol to be successful, the request needs to arrive before the time-out. However, as communication channels can help Bob, this message can be delayed until the time-out occurs. Intuitively, a resilient channel has the capacity to delay messages: thus for each possible finite time-out value chosen by Alice, Bob (with the help of the communication channels) can delay long enough the delivery of the message in order for the TTP to reject the recovery request. The flaw is detected, while verifying the following ATL-formula that evaluates to true.

$$\langle\langle \mathsf{B}, \mathsf{Com} \rangle\rangle\Diamond(\mathsf{EOO} \wedge (\mathsf{EOO_k} \vee \mathsf{Con_k}) \wedge \neg\langle\langle \mathsf{A}, \mathsf{TimeOut} \rangle\rangle\Diamond(\mathsf{EOR} \wedge (\mathsf{EOR_k} \vee \mathsf{Con_k})))$$

The formula verifies whether Bob, collaborating with the communication channels, has a strategy to cheat Alice. However, when we change the communication channel to be operational (messages are transmitted immediately) the protocol becomes fair. Unfortunately, operational channels are rather unrealistic in practice. On the other hand the formula verifying whether Alice has a strategy to cheat Bob fails. This is a good example to illustrate that a CTL-formula trying to express the same property would not be adequate for our purposes. The formula

$$\exists\Diamond(\mathsf{EOR} \wedge (\mathsf{EOR}_k \vee \mathsf{Con}_k) \wedge \neg\exists\Diamond(\mathsf{EOO} \wedge (\mathsf{EOO}_k \vee \mathsf{Con}_k)))$$

evaluates to true, while Alice does not have an attack against Bob in this protocol. It is easy to see that Bob can help Alice to get the non-repudiation of receipt evidence. In the same execution, Alice can abort the protocol after having received Bob's evidences and then stop the protocol, before sending her evidence to Bob. This execution exists, but is discarded in our formalism, as we explicitly express the fact that Bob does not help Alice, as it is the case in the first part of the described execution.

We can also use our method to find the origin of a flaw. We can add players to the cooperation to detect the player(s) that make the property fail. As an example, take a look at the flaw we described in the previous paragraph. We asked whether Bob, together with all the communication channels, has a strategy to obtain the NRO evidence, such that Alice, in cooperation with the clock, does not have any strategy to receive the NRR evidence. Using MOCHA we showed the existence of a flaw, by showing that this formula is true on the ZG protocol. However, it may be difficult to determine the origin of the failure. To find this origin we change the coalitions to find the player(s) responsible for the flaw. By adding the communication channel between Alice and the TTP to Alice's coalition we succeeded in avoiding the property to be validated. This reflected by the fact that the following formula evaluates to false.

$$\langle\langle \mathsf{B}\rangle\rangle\Diamond(\mathsf{EOO} \wedge (\mathsf{EOO_k} \vee \mathsf{Con_k}) \wedge \neg\langle\langle \mathsf{A}, \mathsf{TimeOut}, \mathsf{Com}\rangle\rangle\Diamond(\mathsf{EOR} \wedge (\mathsf{EOR_k} \vee \mathsf{Con_k})))$$

Once we know that the communication channel is at the origin of the error it may be interesting to alter the channel's quality. In that way we see which modifications are needed to recover the fairness property. In this example we can conclude that replacing the resilient channel by an operational channel was enough to prevent protocol failure. Hence, we have a rather easy and quick investigation method to locate and possibly prevent flaws. This methodology may also be useful during the design process of new protocols.

4.2 The ASW certified mail protocol

The ASW certified mail protocol [4] has been designed for use on asynchronous (resilient) networks. The protocol is more complicated: it uses a main protocol, an abort protocol and two recovery protocols. For such complex protocols the use of formal methods is very helpful, as an important number of different situations can occur. Using informal methods Zhou et al. [28] found two problems in the protocol. The first problem compromises fairness for Alice, the second one for Bob. We briefly describe the protocol and therefore introduce some additional notation specific to this protocol.

- key_A and key_B: two fresh random numbers generated by A and B respectively.
- $c = E_{TTP}(m, key_A, A, B)$: the encryption for the TTP of the message m, key_A and the participants' identities.

The main protocol (protocol 3) consists of four messages. In the first message Alice sends a commitment to m, as well as to the non-repudiation of origin evidence to Bob, by sending the hash of the message m, as well as the encryption for the TTP of both the message and the random number key_A. Bob replies by sending a commitment to his random number. Then they go on in messages 3 and 4 by exchanging m and key_A against key_B.

Protocol 3 Main protocol

1. A → B: $m_1 = A, B, TTP, c, h(m), S_A(A, B, TTP, c, h(m))$
2. B → A: $m_2 = h(key_B), S_B(m_1, h(key_B)$
3. A → B: $m_3 = m, key_A$
4. B → A: $m_4 = key_B$

The abort protocol is executed by Alice if Bob does not reply to the first message of the main protocol. When Alice wants to abort the protocol (protocol 4), she sends a signed abort request to the TTP. If at that moment, Bob has already recovered the protocol, the TTP invites Alice to launch a recovery protocol. Otherwise the TTP sends an abort confirmation to Alice.

Protocol 4 Abort protocol

1. A → TTP: $a_1 = aborted, m_1, S_A(aborted, m_1)$

   if B recovered **then** recovery_A **else**
2. TTP → A: $abort\_token = a_1, S_{TTP}(a_1)$

When the third message of the main protocol does not arrive to Bob, he can execute a recovery protocol (protocol 5). Therefore Bob sends a recovery request, including the cipher c to the TTP. If the protocol has already been aborted, the TTP sends the abort token to Bob. Otherwise the TTP decrypts c and sends the message m, as well as key_A to Bob.

Protocol 5 Recovery_B protocol

1. B → TTP: $rb_1 = B, m_1, m_2, key_B$

   if aborted **then**
2. TTP → B: $rb_2 = abort\_token$

   else
3. TTP → B: $rb_3 = m, key_A$

In the same way as Bob, Alice can also recover the protocol (protocol 6) if the fourth message of the main protocol is not arriving. If the protocol has been aborted previously, the TTP sends the abort token. Otherwise an affidavit is issued by the TTP, replacing key_B.

In the protocol, we have that the non-repudiation of origin evidence NRO corresponds to (m_1, key_A) and the non-repudiation of receipt evidence NRR to (m_1, m_2, key_B) or $affidavit\_token$.

When verifying fairness for this protocol, using the techniques described in the previous section, we find that the protocol is neither fair for Alice nor for Bob.

The first problem is due to the fact that Bob can execute his recovery protocol immediately after having received the first message. This has not been foreseen by the protocol: at this moment

Protocol 6 Recovery_A protocol

1. A $\rightarrow$ TTP: $ra_1 = A, m_1, m_2, m, key_A$

   if aborted **then**

2. TTP $\rightarrow$ A: $ra_2 = abort\_token$

   else

3. TTP $\rightarrow$ A: $affidavit\_token = affidavit, ra_1, S_{TTP}(affidavit, ra_1)$

Alice has not enough information to launch her recovery protocol as requested by the TTP and she cannot receive the non-repudiation of receipt evidence for her message. Using our method, we easily detected this flaw: the fact that Alice does not have the ability to execute the recovery protocol is reflected in our specification by a guard that is not enabled.

In the second attack, compromising fairness for Bob, Alice launches a recovery protocol and afterwords the abort protocol. The problem is that in the abort protocol, the TTP only checks whether *Bob* did already recover or not. The TTP does not check whether *Alice* did already recover or not. Hence, a recovery protocol and an abort protocol are not mutually exclusive, if both are executed by Alice, in the given order. It was not foreseen that Alice could launch a recovery protocol followed by an abort protocol. Executing the protocols in this order makes it impossible for Bob to launch his recovery protocol and, hence, breaks fairness. Again, this attack is easily detected by our verification method: the guards of the commands launching the abort protocol and the recovery protocol are both enabled. Hence, Alice can choose the order of execution of the protocols and cheat Bob.

4.3 The KM non-repudiation protocol

The KM non-repudiation protocol overcomes the problems discovered in the two previously discussed protocols. When analyzing the protocol with Mocha we did not find any errors. However, we designed an interesting variant of the protocol where we removed the abort protocol. Removing the abort protocol introduces a new flaw, that does not compromise fairness, as the previous shown errors did, but the timeliness property. We will give a brief informal description of the variant using the notation introduced above. The generated evidences are the following:

- $\mathsf{EOO} = S_A(f_{\mathsf{EOO}}, B, l, h(c))$
- $\mathsf{EOR} = S_B(f_{\mathsf{EOR}}, A, l, h(c))$
- $\mathsf{Sub} = S_A(f_{\mathsf{Sub}}, B, l, E_{TTP}(k))$
- $\mathsf{EOO}_k = S_A(f_{\mathsf{EOO}_k}, B, l, k)$
- $\mathsf{EOR}_k = S_B(f_{\mathsf{EOR}_k}, A, l, k)$
- $\mathsf{Rec}_X = S_X(f_{\mathsf{Rec}_X}, Y, l)$
- $\mathsf{Con}_k = S_{TTP}(f_{\mathsf{Con}_k}, A, B, l, k)$

Protocol 7 Main protocol

1. A $\to$ B: $f_{\mathsf{EOO}}, f_{\mathsf{Sub}}, B, l, c, E_{TTP}(k), \mathsf{EOO}, \mathsf{Sub}$
2. B $\to$ A: $f_{\mathsf{EOR}}, A, l, \mathsf{EOR}$
3. A $\to$ B: $f_{\mathsf{EOO}_k}, B, l, k, \mathsf{EOO}_k$
4. B $\to$ A: $f_{\mathsf{EOR}_k}, A, l, \mathsf{EOR}_k$

In the main protocol we first exchange a cipher of the message under key k against the receipt for this cipher and go on exchanging the key k against a receipt for the key.

The recovery protocol can be launched if a problem in the main protocol occurs. The TTP sends the required evidences to both Alice and Bob.

Protocol 8 Recovery protocol

1. X $\to$ TTP: $f_{\mathsf{Rec}_X}, f_{\mathsf{Sub}}, Y, l, h(c), E_{TTP}(k), \mathsf{Rec}_X, \mathsf{Sub}, \mathsf{EOR}, \mathsf{EOO}$
2. TTP $\to$ A: $f_{\mathsf{Con}_k}, A, B, l, k, \mathsf{Con}_k, \mathsf{EOR}$
3. TTP $\to$ B: $f_{\mathsf{Con}_k}, A, B, l, k, \mathsf{Con}_k$

We can show that this protocol is fair. However, timeliness is not respected as the formula hereunder evaluates to false.

$$\langle\!\langle A \rangle\!\rangle \Diamond(\mathsf{stop}_A \wedge (\neg(\mathsf{EOR} \wedge (\mathsf{EOR}_k \vee \mathsf{Con}_k)) \to \neg\langle\!\langle B \rangle\!\rangle \Diamond(\mathsf{EOO} \wedge (\mathsf{EOO}_k \vee \mathsf{Con}_k))))$$

Imagine the following situation. Alice sends the first message to Bob. At this moment Alice can only wait for message 2 to arrive. Bob, however can either stop the protocol or launch the recovery protocol. In both cases fairness is guaranteed. If Bob waits some time before launching the recovery protocol, Alice does not know whether Bob halted the protocol or whether she should wait for a recovery. Unless Alice is waiting forever, Bob only needs to wait for Alice to quit the protocol and launch the recovery. This attack is due to the fact that Alice has no strategy to finish the protocol in a finite amount of time. This example emphasizes the importance of the timeliness property, that is much less studied than fairness. Our formalism (see formula 13) allows us to formally capture this flaw.

4.4 The MK multi-party non-repudiation protocol

We also studied the multi-party non-repudiation protocol presented in [17]. In this protocol we have one originator, Alice, that sends a message to several recipients. We studied the protocol with only two recipients. In [17] a problem is discussed if the clocks of Alice and the TTP are not synchronized and a solution to the synchronization problem is given. Using Mocha we show that the protocol is unfair for Alice if the synchronization problem is not handled correctly, as proposed in the original paper.

To describe the protocol we have to add the following notation:

- $X \Rightarrow \mathcal{E}$: multicast from entity X to the set of entities $\mathcal{E}$.
- $E_{\mathcal{E}}()$: a group encryption scheme E, that can be deciphered by each party $P \in \mathcal{E}$.

- $\mathcal{B}$: the set of receiving entities.
- $\mathcal{B}'$: the set of receiving entities having sent an evidence of receipt for the cipher to Alice.
- $l = h(m, k)$: a label that in conjunction with the identity A uniquely identifies a protocol run

The generated evidences are the following.

- $\mathsf{EOO} = S_A(f_{\mathsf{EOO}}, \mathcal{B}, l, t, h(c))$
- $\mathsf{EOR}_i = S_{B_i}(f_{\mathsf{EOR}}, A, l, t, h(c))$
- $\mathsf{Sub} = S_A(f_{\mathsf{Sub}}, \mathcal{B}, l, E_{TTP}(k))$
- $\mathsf{EOO}_k = S_A(f_{\mathsf{EOO}_k}, \mathcal{B}', l, h(k))$
- $\mathsf{EOR}_{i,k} = S_{B_i}(f_{\mathsf{EOR}_k}, A, l, h(k))$
- $\mathsf{Rec}_X = S_X(f_{\mathsf{Rec}_X}, A, \mathcal{B}, l)$
- $\mathsf{Con}_k = S_{TTP}(f_{\mathsf{Con}_k}, A, \mathcal{B}', l, h(k))$
- $\mathsf{Early} = S_{TTP}(f_{\mathsf{Early}}, l)$
- $\mathsf{Set} = S_A(f_{\mathsf{Set}}, \mathcal{B}', l)$

In the main protocol Alice sends a cipher of the message to all potential receivers $\mathcal{B}$. Several of these receivers $\mathcal{B}' \subseteq \mathcal{B}$ are sending a receipt for this cipher. Alice continues the protocol with the receivers in $\mathcal{B}'$ by sending them the key corresponding to the cipher of message 1.

Protocol 9 Main protocol

1. A $\Rightarrow \mathcal{B}$: $f_{\mathsf{EOO}}, f_{\mathsf{Sub}}, \mathcal{B}, l, t, c, E_{TTP}(k), \mathsf{EOO}, \mathsf{Sub}$
2. $sfB_i \rightarrow$ A: $f_{\mathsf{EOR}}, A, l, \mathsf{EOR}_i$

   where $\mathsf{B_i} \in \mathcal{B}$ and $i \in \{1, \ldots, |\mathcal{B}|\}$
3. A $\Rightarrow \mathcal{B}'$: $f_{\mathsf{EOO}_k}, \mathcal{B}', l, E_{\mathcal{B}'}(k), \mathsf{EOO}_k$
4. $\mathsf{B'_j} \rightarrow$ A: $f_{\mathsf{EOR}_k}, A, l, \mathsf{EOR}_{j,k}$

   where $\mathsf{B'_j} \in \mathcal{B}'$ and $j \in \{1, \ldots, |\mathcal{B}'|\}$

The first message destined to all receivers in $\mathcal{B}$ includes the label, the cipher, as well as the key k ciphered using the public key of the TTP (this information is used by the TTP in the case of a recovery). Alice also sends a time-out t: a recovery may only be initiated after t. If one of the receivers does not accept the time-out he may stop the protocol.

After having sent the cipher in message 1, Alice decides of the moment to continue the protocol. All receipts arriving after this moment are not considered any more, without any risk for the receivers of losing fairness.

Afterwords, when Alice sends the deciphering key it is crucial that only the recipients in $\mathcal{B}'$ receive it. Therefore we need to use a cipher (this is not needed in an optimistic two-party protocol).

In order to cipher only once and to use multicasting, Alice uses a group encryption scheme, e.g. [10].

At each moment during the main protocol Alice and Bob have the possibility to launch the recovery protocol with the TTP. The recipients in $\mathcal{B}'$ launch a recovery if Alice does not multicast the ciphered key; Alice initiates the recovery protocol, if not all recipients in $\mathcal{B}'$ send a receipt for the ciphered key.

Protocol 10 Recovery protocol

1. X $\rightarrow$ TTP: $f_{\mathsf{Rec}_X}, f_{\mathsf{Sub}}, A, \mathcal{B}, l, t, h(c), E_{TTP}(k), \mathsf{Rec}_X, \mathsf{Sub}, \mathsf{EOO}$

   if recovered then

2. TTP $\rightarrow$ X: $f_{\mathsf{Con}_k}, A, \mathcal{B}', l, E_{\mathcal{B}'}(k, S_{TTP}(k)), \mathsf{Con}_k$

   else if before t then

3. TTP $\rightarrow$ X: $f_{\mathsf{Early}}, \mathsf{Early}$

   else

   recovered=true

4. TTP $\leftrightarrow$ A: $f_{\mathsf{Set}}, \mathcal{B}', l, \mathsf{Set}$
5. TTP $\rightarrow$ A: $f_{\mathsf{Con}_k}, A, \mathcal{B}', l, \mathsf{Con}_k$
6. TTP $\Rightarrow$ $\mathcal{B}' \cup \{\mathsf{X}\} \backslash \{\mathsf{A}\}$: $f_{\mathsf{Con}_k}, A, \mathcal{B}', l, E_{\mathcal{B}'}(k, S_{TTP}(k)), \mathsf{Con}_k$

Before t (t is specified by Alice during the first message in the main protocol and it is resent together with Alice's signature during the recovery request) the recovery cannot be initiated. If someone tries to execute a recovery before t, which is resent together with Alice's signature during the recovery request, the TTP sends a message to notify that the recovery cannot yet be initiated.

When the recovery protocol is initiated the first time after t, the TTP uses ftp get to fetch the set $\mathcal{B}'$ at Alice. For this purpose Alice maintains a read-only publicly accessible directory containing the set of users in $\mathcal{B}'$ and her signature on this set. If the directory is not accessible or empty the TTP supposes $\mathcal{B}' = \emptyset$.

Now the TTP sends to Alice the confirmation of receipt of the key that may be used to substitute $\mathsf{EOR}_{i,k}$ for all i such that $B_i \in \mathcal{B}'$. The TTP sends to all receiver $B_i \in \mathcal{B}'$ the confirmation of the key, that substitutes EOO_k, as well as the signed key ciphered for $\mathcal{B}'$ using a group encryption scheme.

If a receiver $B_i \notin \mathcal{B}'$ wants to perform a recovery, after the recovery has already been performed for the first time, the TTP uniquely identifies each protocol run by l and A, the TTP sends to this entity the same message as to each $B_i \in \mathcal{B}'$. This message is however useless as k has been ciphered for the set $\mathcal{B}'$ and only informs the recipient that he does not belong to this set.

Using MOCHA we can show that this version of the protocol contains a subtle flaw, due to a possible shift between the TTP's clock and Alice's clock, associated to a race in the transmissions. In order to show this we verify that both B_1 and B_2 can collaborate to break fairness for Alice. Without loss of generality we suppose that it is B_1, who gets the advantage. The ATL-formula we verify is the following :

$$\langle\langle B_1, B_2, \mathsf{Com}, \mathsf{TimeOut}\rangle\rangle \mathsf{NRO}_1 \wedge \neg\langle\langle \mathsf{A}\rangle\rangle \mathsf{NRR}_1$$

NRO_1 and NRR_1 represent the complete non-repudiation of origin, respectively receipt evidences related to B_1. $\mathsf{TimeOut}$ is a player that controls the clocks of Alice and the TTP, and has to respect fairness conditions on the triggering of the time-outs. The flawed scenario is as follows.

Suppose that the clock of the TTP indicates that the moment t arrived whereas it is not yet the case on the clock of Alice. B_1 can initiate the recovery protocol without having sent the the second message of the main protocol. The TTP will consult Alice's public directory and will get a set $\mathcal{B}'$ not containing B_1. If B_1 now sends the second message of the main protocol before the moment t on the clock of Alice and before the third message of the recovery protocol, sent by the TTP, arrives to Alice, Alice will include B_1 in $\mathcal{B}'$. $\mathcal{B}'$ will be different from the one obtained by the TTP. If Alice sends the third message of the main protocol, before receiving the message sent by the TTP, B_1 possesses the message m and the non-repudiation of origin evidence. If B_1 doesn't send the last message of the main protocol, Alice will receive from the TTP a non-repudiation of receipt evidence for a set of recipients that only contains B_2. Hence, fairness is broken.

In order to avoid this flaw, Alice must stop taking part at the main protocol as soon as the TTP consults her public directory. Using this additional constraint, we did not detect any additional flaw. Note, that currently we are only able to verify the protocol with 2 Bs, respectively a fixed number of Bs. A more general method allowing arbitrary number of Bs is left for future works.

5 Related Works

First efforts to apply formal methods to the verification of non-repudiation protocols have been presented by Zhou et al. in [29]. SVO, a "BAN-like" belief logic has been used to study a non-repudiation protocol. The aim of that study differs however from our study. In the context of non-repudiation protocols, belief logics are useful to reason about the validity of the evidences. They deal with questions of what a judge has to believe when Alice or Bob present their respective evidences. Belief logics cannot be applied to verify properties such as fairness or timeliness. In [22] Schneider uses CSP to prove the correctness of a non-repudiation protocol. These proofs are not automated and require great efforts. Recently Boyd and Kearny [8] discussed a method using specification animation to analyse fair exchange protocols using the Possum animation tool. The tool gives the possibility to step through the protocol and examine the consequences of various actions. Boyd et al. use a high level abstraction that only allows to find errors due to sending messages out of order. However they succeed in finding errors even on these very simplified versions of the protocols.

The most extensive studies of fair exchange protocols using formal methods have been presented by Shmatikov and Mitchell in [23, 24, 25]. They use Murφ, a finite-state model-checker to analyze a fair exchange and two contract signing protocols. Their approach differs from our approach on some major points. First, they use an intruder model. To model the fact that a party is malicious and could try to cheat, they make that party share all its knowledge with the intruder. The same intruder model has been used to analyse other security protocols. However we believe that there is no need to use an intruder model. Therefore we directly model Alice as well as Bob to be potentially malicious. The analysis using Murφ is based on invariant checking. Channel resilience is obtained by checking all invariants in final protocol states. Protocol runs where messages are lost do not reach the final protocol states and are discarded. To achieve resilience in our model we need to add fairness conditions. Invariant checking is sufficient to verify fairness as fairness is a *monotonic* property: if fairness is broken at one point of the protocol, the protocol will remain unfair. Checking only final states is thus sufficient to verify fairness. However with this approach timeliness cannot be verified. Timeliness is a liveness property guaranteeing that we can always reach a final state in the protocol. A protocol run, not respecting timeliness, would be discarded for the invariant check as the protocol would not be finished. Note that Shmatikov et al. did not aim to verify timeliness. In our model the timeliness property can easily be verified. As shown before, timeliness is of crucial importance for the security of a protocol and should be verified.

6 Conclusions and Future Works

We have shown that exchange protocols, due to their particularities, are best modeled as games. First, the adversarial and cooperative behaviors that occur during the execution of those protocols are naturally and precisely expressed in term of strategies. Second, the main requirements that an exchange protocol must ensure are easily phrased as existence of strategies for the participating entities to reach well defined goals. We have proposed to use the framework of alternating transition systems and alternating-time temporal logic to formalize this view of exchange protocols as games. The great expressive power of alternating temporal logic in this context has been illustrated. The practical interest of the method has been demonstrated by analyzing several protocols. For the analysis, we have used the tool MOCHA which is able to automatically verify alternating temporal formula on alternating transition systems.

In the future, we will try to show that this approach is applicable to the analysis and verification of more specific properties, such as abuse-freeness in contract signing protocols for instance. We will also investigate the problem of strategy synthesis. Strategy synthesis is a very powerful tool in protocol design and can also be used to synthesize attacks, showing the origin of flaws.

Acknowledgement. The authors would like to thank Giorgio Delzanno, Olivier Markowitch and Thierry Massart for carefully reading previous versions of this paper, as well as the members of the group "Vérification" of ULB, FUNDP and UMH for their comments on that work. Many thanks to Freddy Mang for his assistance while using MOCHA and to the anonymous referees for their helpful comments.

References

[1] R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, 1997.

[2] R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: modularity in model checking. In A. Hu and M. Vardi, editors, *CAV 98: Computer-aided Verification*, Lecture Notes in Computer Science 1427, pages 521–525. Springer-Verlag, 1998.

[3] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In T. Matsumoto, editor, *4th ACM Conference on Computer and Communications Security*, pages 6, 8–17, Zurich, Switzerland, Apr. 1997. ACM Press.

[4] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, Oakland, CA, May 1998. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[5] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606, 1998.

[6] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[7] D. Boneh and M. Naor. Timed commitments. In M. Bellare, editor, *Advances in Cryptology: Proceedings of Crypto 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer-Verlag, 2000.

[8] C. Boyd and P. Kearney. Exploring fair exchange protocols using specification animation. In *The Third International Workshop on Information Security - ISW2000*, Lecture Notes in Computer Science, Australia, Dec. 2000. Springer-Verlag.

[9] M. Burrows, M. Abadi, and R. Needham. A logic of authentication, from proceedings of the royal society, volume 426, number 1871, 1989. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996.* 1996.

[10] G. Chiou and W. Chen. Secure broadcasting using the secure lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, Aug. 1989.

[11] S. Even and Y. Yacobi. Relations among public key signature systems. Technical Report 175, Technion, Haifa, Israel, Mar. 1980.

[12] T. Henzinger, R. Manjumdar, F. Mang, and J.-F. Raskin. Abstract interpretation of game properties. In *SAS 2000: Intertional Symposium on Static Analysis*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[13] S. Kremer and O. Markowitch. Optimistic non-repudiable information exchange. In J. Biemond, editor, *21th Symp. on Information Theory in the Benelux*, pages 139–146. Werkgemeenschap Informatie- en Communicatietheorie, Enschede, may 2000.

[14] S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In K. Larsen and M. Nielsen, editors, *CONCUR: 12th International Conference on Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, Aalborg, Denmark, Aug. 2001. Springer-Verlag.

[15] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, Nov. 1995.

[16] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[17] O. Markowitch and S. Kremer. A multi-party optimistic non-repudiation protocol. In D. Won, editor, *Proceedings of The 3rd International Conference on Information Security and Cryptology (ICISC 2000)*, volume 2015 of *Lecture Notes in Computer Science*, Seoul, Korea, 2000. Springer-Verlag.

[18] O. Markowitch and Y. Roggeman. Probabilistic non-repudiation without trusted third party. In *Second Conference on Security in Communication Networks'99*, Amalfi, Italy, Sept. 1999.

[19] C. A. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. *Lecture Notes in Computer Science*, 1146:351–365, 1996.

[20] S. Micali. Certified E-mail with invisible post offices. Available from author; an invited presentation at the RSA '97 conference, 1997.

[21] L. C. Paulson. Proving properties of security protocols by induction. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, June 1997.

[22] S. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW '98)*, pages 54–65, Washington - Brussels - Tokyo, June 1998. IEEE.

[23] V. Shmatikov and J. Mitchell. Analysis of a fair exchange protocol. In *Symposium on Network and Distributed Systems Security (NDSS '00)*, pages 119–128, San Diego, CA, Feb. 2000. Internet Society.

[24] V. Shmatikov and J. Mitchell. Analysis of abuse-free contract signing. In *Financial Cryptography '00*, volume 1962 of *Lecture Notes in Computer Science*, Anguilla, Feb. 2000. Springer-Verlag.

[25] V. Shmatikov and J. Mitchell. Finite-state analysis of two contract signing protocols. *Special issue of Theoretical Computer Science on security*, 2001. Accepted for publication.

[26] T. Tedrick. How to exchange half a bit. In D. Chaum, editor, *Advances in Cryptology: Proceedings of Crypto 83*, pages 147–151. Plenum Press, New York and London, 1984, 22–24 Aug. 1983.

[27] T. Tedrick. Fair exchange of secrets. In G. R. Blakley and D. C. Chaum, editors, *Advances in Cryptology: Proceedings of Crypto 84*, volume 196 of *Lecture Notes in Computer Science*, pages 434–438. Springer-Verlag, 1985.

[28] J. Zhou, R. Deng, and F. Bao. Some remarks on a fair exchange protocol. In *2000 International Workshop on Practice and Theory in Public Key Cryptography*, Lecture Notes in Computer Science, Melbourne, Australia, Jan. 2000. Springer-Verlag.

[29] J. Zhou and D. Gollmann. Towards verification of non-repudiation protocols. In *Proceedings of 1998 International Refinement Workshop and Formal Methods Pacific*, pages 370–380, Canberra, Australia, Sept. 1998. Springer-Verlag.

[30] Y. Zhou and D. Gollmann. An efficient non-repudiation protocol. In *Proceedings of The 10th Computer Security Foundations Workshop*, pages 126–132. IEEE Computer Society Press, June 1997.