Implementing a Constraint Solving Algorithm for Checking Game-Theoretic Security Requirements

Ralf Küsters, Thomas Schmidt, and Tomasz Truderung

University of Trier, Germany

Abstract. Contract-signing and related protocols have to satisfy game-theoretic security properties. Based on an algorithm proposed by Kähler and Küsters for checking such security properties, in this work we report on the implementation of this algorithm and its performance.

1 Introduction

One of the central results in the area of automatic analysis of cryptographic protocols is that the security of cryptographic protocols is decidable when analyzed w.r.t. a finite number of sessions, without a bound on the message size, and in presence of the so-called Dolev-Yao intruder (see, e.g., [17, 1]). Based on this result, many fully automatic tools (see, e.g., [2, 9, 16]) have been developed and successfully applied to find flaws in published protocols, where many of these tools employ so-called *constraint solving procedures* (see, e.g., [16, 9, 5]). However, the mentioned decidability result and tools are restricted to security properties such as authentication and secrecy which are reachability properties of the transition system associated with a given protocol.

In contrast, crucial properties required of contract-signing and related protocols (see, e.g., [11, 4]), for instance abuse-freeness [11] and balance [6], are game-theoretic properties of the structure of the transition system associated with a protocol. Balance, for instance, requires that in no stage of a protocol run, the intruder or a dishonest party has both a strategy to abort the run and a strategy to successfully complete the run and thus obtain a valid contract.

In [13], the central decidability result mentioned above was extended to such game-theoretic security properties, including balance. However, similar to the result by Rusinowitch and Turuani [17] for reachability properties, the decision algorithm presented in [13] is merely based on the fact that the size of attacks can be bounded, and hence, all potential attacks up to a certain size have to be enumerated and checked. In [12, 14], a *constraint-based* decision procedure for game-theoretic security properties of the kind considered in [13] was proposed. The main feature of this procedure is that it can be built on top of *standard constraint solving procedures* (see, e.g., [16, 9, 5] and references therein). As mentioned, such procedures have successfully been employed for reachability properties in the past and proved to be a good basis for practical implementations. Hence, the constraint-based procedure presented in [12, 14] appeared to be a promising basis for extending existing implementations and tools for reachability properties to deal with game-theoretic security properties.

The main goal of this work is to turn the algorithm proposed in [12, 14] into a practical implementation. This is a challenging task for several reasons: First, protocols which try to satisfy game-theoretic security properties, such as contract-signing protocols, tend to be quite complex. For instance, the ASW contract-signing protocol [4] consists of four sub-protocols, which, in addition, have a branching structure, instead of a simpler linear structure, as common, for example, for authentication and key exchange protocols. This leads to a bigger number of constraint systems to be checked. Second, while for secrecy and authentication it suffices to analyze single protocol runs, for game-theoretic security requirement more complex strategy trees need to be considered, leading to complex constraint systems.

Contribution of this paper. We implemented the algorithm proposed in [12, 14] in Prolog based on the constraint solver by Corin and Etalle [10] (which in turn is based on the constraint solver by Millen and Shmatikov [16]). Our implementation contains several optimizations. We applied our implementation to check various balance properties for, among others, the ASW protocol. It turned out that the number of generated constraint systems, although indeed quite big, was not a significant obstacle in the verification process. More problematic was the size of the single constraint systems, which, for the ASW protocol, overwhelmed the constraint solver we used. For this protocol, we therefore introduced some sound simplifications, which reduced the size of the terms in the generated constraint systems. With these simplifications the verification process was successful.

Altogether, our implementation is promising in that it shows that, in principle, automatic analysis of game-theoretic security properties is possible for non-trivial contract-signing protocols. However, more work is necessary to optimize our implementation. Moreover, while we used the constraint solver by Corin and Etalle [10] since it could be easily integrated into our Prolog implementation, more efficient constraint solvers, such as those in [3], should yield better results.

Structure of this paper. In Section 2 we recall definitions and results from [12, 14]. Our implementation is described in Section 3, with the evaluation presented in Section 4. We conclude in Section 5.

2 Constraint Solving Algorithm for Contract-Signing Protocols

In this section we recall definitions and results from [12, 14].

2.1 The Protocol and Intruder Model

In the model proposed in [12, 14], a protocol is a finite set of principals and every principal is a finite tree, which represents all possible behaviors of the principal, including all subprotocols a principal can carry out. Each edge of such a tree is labeled by a rewrite rule, which describes the receive-send action that is performed when the principal takes this edge in a run of the protocol.

When a principal carries out a protocol, it traverses its tree, starting at the root. In every node, the principal takes its current input, chooses one of the edges leaving

the node, matches the current input with the left-hand side of the rule the edge is labeled with, sends out the message which is determined by the right-hand side of the rule, and moves to the node the chosen edge leads to. While in the standard Dolev-Yao model (see, e.g., [17]) inputs to principals are always provided by the intruder, in this model inputs can also come from the secure channel which are not controlled by the intruder, i.e., the intruder cannot delay, duplicate, remove messages, or write messages onto this channel under a fake identity (unless he has corrupted a party).

Terms and Messages. As usual, we have a finite set $\mathcal{V}$ of variables, a finite set $\mathcal{A}$ of atoms, a finite set $\mathcal{K}$ of public and private keys equipped with a bijection $\cdot^{-1}$ assigning public to private keys and vice versa. In addition, we have a finite set $\mathcal{N}$ of *principal addresses* for the secure channels and an *infinite* set $\mathcal{A}_I$ of *intruder atoms*, containing nonces and symmetric keys the intruder can generate. All of the mentioned sets are assumed to be disjoint.

We define two kinds of terms by the following grammar, namely *plain terms* and *secure channel terms*:

$$\begin{aligned} \textit{plain-terms} ::= {} & \mathcal{V} \mid \mathcal{A} \mid \mathcal{A}_I \mid \langle \textit{plain-terms}, \textit{plain-terms} \rangle \mid \{\textit{plain-terms}\}^{s}_{\textit{plain-terms}} \mid \\ & \{\textit{plain-terms}\}^{a}_{\mathcal{K}} \mid \mathsf{hash}(\textit{plain-terms}) \mid \mathsf{sig}_{\mathcal{K}}(\textit{plain-terms}) \\ \textit{sec-terms} ::= {} & \mathsf{sc}(\mathcal{N}, \mathcal{N}, \textit{plain-terms}) \\ \textit{terms} ::= {} & \textit{plain-terms} \mid \textit{sec-terms} \mid \mathcal{N} \end{aligned}$$

While the plain terms are standard in Dolev-Yao models, a secure channel term of the form $\mathsf{sc}(n, n', t)$ stands for feeding the secure channel from n to n' with t. Knowing n grants access to secure channels with sender address n. A *(plain/secure channel) message* is a (plain/secure channel) ground term, i.e., a term without variables.

Intruder. Given a set $\mathcal{I}$ of messages, the (infinite) set $d(\mathcal{I})$ of messages the intruder can derive from $\mathcal{I}$ is defined in a standard way (see [14]) with the additional rule handling secure channels: if $m \in d(\mathcal{I})$, $n \in d(\mathcal{I}) \cap \mathcal{N}$, and $n' \in \mathcal{N}$, then $\mathsf{sc}(n, n', m) \in d(\mathcal{I})$ (*writing onto the secure channel*). Intuitively, $n \in d(\mathcal{I}) \cap \mathcal{N}$ means that the intruder has corrupted the principal with address n and therefore can impersonate this principal when writing onto the secure channel.

In this model, all dishonest parties are subsumed by the intruder.

Principals and Protocols. Principal rules are of the form $L \Rightarrow R$ where L is a term or ε and R is a term.

A *rule tree* $\Pi = (V, E, r, \ell)$ is a finite tree rooted at $r \in V$ where ℓ maps every edge $(v, v') \in E$ of Π to a principal rule $\ell(v, v')$. A *principal* is a rule tree that satisfies some additional conditions (namely well-formedness and feasibility; see [14] for details).

For $v \in V$, we write $\Pi{\downarrow}v$ to denote the subtree of Π rooted at v. For a substitution σ, we write $\Pi\sigma$ for the principal obtained from Π by substituting all variables x occurring in the principal rules of Π by $\sigma(x)$.

A *protocol* $P = ((\Pi_1, \ldots, \Pi_n), \mathscr{I})$ consists of a finite sequence of principals Π_i and a finite set $\mathscr{I}$ of messages, the *initial intruder knowledge*. We require that each variable occurs in the rules of only one principal, i.e., different principals must have disjoint sets of variables. We assume that intruder atoms, i.e., elements of $\mathscr{A}_I$, do not occur in P.

Transition Graph Induced by a Protocol A transition graph $\mathscr{G}_P$ induced by a protocol P comprises all runs of a protocol. To define this graph, we first introduce states and transitions between these states.

A *state* is of the form $((\Pi_1, \ldots, \Pi_n), \sigma, \mathscr{I}, \mathscr{S})$ where σ is a ground substitution, for each i, Π_i is a rule tree such that $\Pi_i\sigma$ is a principal, $\mathscr{I}$ is a finite set of messages, the *intruder knowledge*, and $\mathscr{S}$ is a finite multi-set of secure channel messages, the *secure channel*. The idea is that when the transition system gets to such a state, then the substitution σ has been performed, the accumulated intruder knowledge is what can be derived from $\mathscr{I}$, the secure channels hold the messages in $\mathscr{S}$, and for each i, Π_i is the "remaining protocol" to be carried out by principal i. This also explains why $\mathscr{S}$ is a multi-set: messages sent several times should be delivered several times. Given a protocol $P = ((\Pi_1, \ldots, \Pi_n), \mathscr{I})$ the *initial state of* P is $((\Pi_1, \ldots, \Pi_n), \sigma, \mathscr{I}, \emptyset)$ where σ is the substitution with empty domain.

There are three kinds of transitions: intruder, secure channel, and ε-transitions. In what follows, let $\Pi_i = (V_i, E_i, r_i, \ell_i)$ and $\Pi_i' = (V_i', E_i', r_i', \ell_i')$ denote rule trees. A transition

$$((\Pi_1, \ldots, \Pi_n), \sigma, \mathscr{I}, \mathscr{S}) \xrightarrow{\tau} ((\Pi_1', \ldots, \Pi_n'), \sigma', \mathscr{I}', \mathscr{S}') \tag{1}$$

with label τ exists if one of the three following conditions is satisfied:

1. *Intruder transition,* $\tau = [i, m, I]$: There exists a vertex $v \in V_i$ such that $(r_i, v) \in E_i$ and $\ell_i(r_i, v) = R \Rightarrow S$, and there exists a ground substitution σ'' with $\mathsf{dom}(\sigma'') \subseteq \mathscr{V}(R\sigma)$ such that
   (a) $m \in d(\mathscr{I})$,
   (b) $\sigma' = \sigma \cup \sigma''$,
   (c) $R\sigma' = m$,
   (d) $\Pi_j' = \Pi_j$ for every $j \neq i$, $\Pi_i' = \Pi_i{\downarrow}v$,
   (e) either $S \in \mathscr{T}_{\mathsf{plain}}$ and $\mathscr{I}' = \mathscr{I} \cup \{S\sigma'\}$, or $S = \mathsf{sc}(\cdot, \cdot, t)$ for some term t and $\mathscr{I}' = \mathscr{I} \cup \{t\sigma'\}$, and
   (f) either $S \in \mathscr{T}_{\mathsf{plain}}$ and $\mathscr{S}' = \mathscr{S}$, or $S \in \mathscr{T}_{\mathsf{sc}}$ and $\mathscr{S}' = \mathscr{S} \cup \{S\sigma'\}$.
   This transition models that principal i receives the message m from the intruder (i.e., from the public network). Note that the second clause in (e) accounts for the fact that our secure channels are not read-protected.
2. *Secure channel transition,* $\tau = [i, m, \mathsf{sc}]$: There exists a vertex $v \in V_i$ with $(r_i, v) \in E_i$ and $\ell_i(r_i, v) = R \Rightarrow S$, and there exists a ground substitution σ'' with $\mathsf{dom}(\sigma'') \subseteq \mathscr{V}(R\sigma)$ such that (a) $m \in \mathscr{S}$, (b)–(e) as above, and (f) either $S \in \mathscr{T}_{\mathsf{plain}}$ and $\mathscr{S}' = \mathscr{S} \setminus \{m\}$, or $S \in \mathscr{T}_{\mathsf{sc}}$ and $\mathscr{S}' = (\mathscr{S} \setminus \{m\}) \cup \{S\sigma'\}$. This transition models that principal i reads message m from a secure channel.

3. *ε-transition,* $\tau = [i]$: There exists a vertex $v \in V_i$ with $(r_i, v) \in E_i$ and $\ell_i(r_i, v) = \varepsilon \Rightarrow S$ such that $\sigma' = \sigma$ and (d)–(f) as given above for intruder transitions. This transition models that principal i performs a step where neither a message is read from the intruder nor from a secure channel.

Given a protocol P, the *transition graph* $\mathcal{G}_P$ *induced by* P is the tuple (S_P, E_P, q_P) where q_P is the initial state of P, S_P is the set of states reachable from q_P by a sequence of transitions, and E_P is the set of all transitions among states in S_P. We write $q \in \mathcal{G}_P$ if q is a state in $\mathcal{G}_P$ and $q \xrightarrow{\tau} q' \in \mathcal{G}_P$ if $q \xrightarrow{\tau} q'$ is a transition in $\mathcal{G}_P$.

2.2 Intruder Strategies and Strategy Properties

We now define intruder strategies on transition graphs and the goal the intruder tries to achieve following his strategy. To define intruder strategies, we introduce the notion of a strategy tree, which captures that the intruder has a way of acting such that regardless of how the other principals act he achieves a certain goal, where goal in our context means that a state will be reached where the intruder can derive certain constants and cannot derive others (e.g., for balance, the intruder tries to obtain IntruderHasContract but tries to prevent HonestPartyHasContract from occurring).

Definition 1. *For* $q \in \mathcal{G}_P$ *a* q-strategy tree $\mathcal{T}_q = (V, E, r, \ell_V, \ell_E)$ *is an unordered tree where every vertex* $v \in V$ *is mapped to a state* $\ell_V(v) \in \mathcal{G}_P$ *and every edge* $(v, v') \in E$ *is mapped to a label of a transition such that the following conditions are satisfied for all* $v, v' \in V$*, principals* j*, messages* m*, and states* q', q''*:*

1. $\ell_V(r) = q$.
2. $\ell_V(v) \xrightarrow{\ell_E(v,v')} \ell_V(v') \in \mathcal{G}_P$ *for all* $(v, v') \in E$*. (Edges correspond to transitions.)*
3. *If* $\ell_V(v) = q'$ *and* $q' \xrightarrow{j} q'' \in \mathcal{G}_P$*, then there exists* $v'' \in V$ *such that* $(v, v'') \in E$*,* $\ell_V(v'') = q''$*, and* $\ell_E(v, v'') = j$*. (All ε-transitions originating in* q' *must be present in* $\mathcal{T}_q$*.)*
4. *If* $\ell_V(v) = q'$ *and* $q' \xrightarrow{j,m,\mathsf{sc}} q'' \in \mathcal{G}_P$*, then there exists* $v'' \in V$ *such that* $(v, v'') \in E$*,* $\ell_V(v'') = q''$*, and* $\ell_E(v, v'') = j, m, \mathsf{sc}$*. (The same as 3. for secure channel transitions.)*
5. *If* $(v, v') \in E$*,* $\ell_E(v, v') = j, m, I$*, and there exists* $q'' \neq \ell_V(v')$ *with* $\ell_V(v) \xrightarrow{j,m,I} q'' \in \mathcal{G}_P$*, then there exists* v'' *with* $(v, v'') \in E$*,* $\ell_E(v, v'') = j, m, I$ *and* $\ell_V(v'') = q''$*. (The intruder cannot choose which principal rule is taken by* j *if several are possible given the input provided by the intruder.)*

A *strategy property*, i.e., the goal the intruder tries to achieve, is a tuple $((C_1, C_1'), \ldots, (C_l, C_l'))$ where $C_i, C_i' \subseteq \mathscr{A} \cup \mathscr{K} \cup \mathscr{N}$. A state $q \in \mathcal{G}_P$ satisfies $((C_1, C_1'), \ldots, (C_l, C_l'))$ if there exist q-strategy trees $\mathcal{T}_1, \ldots, \mathcal{T}_l$ such that every $\mathcal{T}_i$ satisfies (C_i, C_i') where $\mathcal{T}_i$ satisfies (C_i, C_i') if for all leaves v of $\mathcal{T}_i$ all elements from C_i can be derived by the intruder and all elements from C_i' cannot, i.e., $C_i \subseteq d(\mathscr{I})$ and $C_i' \cap d(\mathscr{I}) = \emptyset$ where $\mathscr{I}$ denotes the intruder knowledge in state $\ell_V(v)$.

The decision problem STRATEGY asks, given a protocol P and a strategy property $((C_1, C_1'), \ldots, (C_l, C_l'))$, whether there exists a state $q \in \mathscr{G}_P$ that satisfies the property. In this case we write $(P, (C_1, C_1'), \ldots, (C_l, C_l')) \in$ STRATEGY.

Note that in a q-strategy tree $\mathscr{T}_q$ there may exist vertices $v' \neq v$ with $\ell_V(v') = \ell_V(v)$ such that the subtrees $\mathscr{T}_q{\downarrow}v$ and $\mathscr{T}_q{\downarrow}v'$ of $\mathscr{T}_q$ rooted at v and v', respectively, are not isomorphic. In other words, the intruder's strategy may depend on the path that leads to a state (i.e., the history) rather than on the state alone, as is the case for positional strategies. We note that the strategies defined in [13] are positional. However, it is easy to see that in our setting both notions of strategies are equivalent. The motivation for using history dependent strategies is that the constraint-based algorithm (Section 2.4) becomes considerably simpler.

2.3 Constraint Solving

In this section, we introduce constraint systems and state the well-known fact that procedures for solving these systems exist (see, e.g., [16, 14] for more details). In Section 2.4, we will then use such a procedure as a black-box for our constraint-based algorithm.

A *constraint* is of the form $t : T$ where t is a plain term and T is a finite non-empty set of plain terms. Since we will take care of secure channel terms when turning the symbolic branching structure into a constraint system, we can disallow secure channel terms in constraints.

A *constraint system* $\mathbf{C}$ is a tuple consisting of a sequence $s = t_1 : T_1, \ldots, t_n : T_n$ of constraints and a substitution τ such that i) the domain of τ is disjoint from the set of variables occurring in s and, ii) for all x in the domain of τ, $\tau(x)$ only contains variables also occurring in s. We call $\mathbf{C}$ *simple* if t_i is a variable for all i. We call $\mathbf{C}$ *valid* if it satisfies the origination and monotonicity property as defined in [16]. The precise definition of valid constraint systems is not needed for the rest of the paper. Let us only note that origination and monotonicity are standard restrictions on constraint systems imposed by constraint solving procedures. Valid constraint systems are all that is needed in our setting.

A ground substitution σ where the domain of σ is the set of variables in $t_1 : T_1, \ldots, t_n : T_n$ is a *solution* of $\mathbf{C}$ ($\sigma \vdash \mathbf{C}$) if $t_i\sigma \in d(T_i\sigma)$ for every i. We call $\sigma \circ \tau$ (the composition of σ and τ read from right to left) a *complete* solution of $\mathbf{C}$ ($\sigma \circ \tau \vdash_c \mathbf{C}$) with τ as above.

A simple constraint system $\mathbf{C}$ obviously has a solution. One such solution, which we denote by σ_C, replaces all variables in $\mathbf{C}$ by new intruder atoms $a \in \mathscr{A}_I$ where different variables are replaced by different atoms. We call σ_C the *solution associated with* $\boldsymbol{C}$ and $\sigma_C \circ \tau$ the *complete solution associated with* $\boldsymbol{C}$.

Given a constraint system $\mathbf{C}$, a finite set $\{\mathbf{C}_1, \ldots, \mathbf{C}_n\}$ of simple constraint systems is called a *sound and complete solution set for* $\boldsymbol{C}$ if $\{\nu \mid \nu \vdash_c \mathbf{C}\} = \{\nu \mid \exists i \text{ s.t. } \nu \vdash_c \mathbf{C}_i\}$. Note that $\mathbf{C}$ does not have a solution iff $n = 0$.

From results shown, for example, in [9, 16, 5] it follows:

Fact 1. *There exists a procedure which given a valid constraint system* $\boldsymbol{C}$ *outputs a sound and complete solution set for* $\boldsymbol{C}$.

While different constraint solving procedures (and implementations thereof) may compute different sound and complete solution sets, our constraint-based algorithm introduced in Section 2.4 works with any of these procedures. It is only important that the set computed is sound and complete. As already mentioned in the introduction, to decide reachability properties it suffices if the procedure only returns one simple constraint system in the sound and complete set. However, the constraint solving procedures proposed in the literature are typically capable of returning a sound and complete solution set.

In what follows, we fix one such procedure and call it the *constraint solver*. More precisely, w.l.o.g., we consider the constraint solver to be a non-deterministic algorithm which non-deterministically chooses a simple constraint system from the sound and complete solution set and returns this system as output. We require that for every simple constraint system in the sound and complete solution set, there is a run of the constraint solver that returns this system. If the sound and complete set is empty, the constraint solver always returns no.

2.4 The Constraint-Based Algorithm

We now present our constraint-based algorithm, called SolveStrategy, for deciding STRATEGY. As mentioned, it uses a standard constraint solver (Fact 1) as a subprocedure.

In what follows, we present the main steps performed by SolveStrategy, with more details given below. The input to SolveStrategy is a protocol P and a strategy property $((C_1, C_1'), \ldots, (C_l, C_l'))$.

1. Guess a symbolic branching structure $\mathbf{B}$, i.e., guess a symbolic path π^s from the initial state of P to a symbolic state q^s and a symbolic q^s-strategy tree $\mathscr{T}^s_{i,q^s}$ for every (C_i, C_i') starting from this state (see below for more details).
2. Derive from $\mathbf{B} = \pi^s, \mathscr{T}^s_{1,q^s}, \ldots, \mathscr{T}^s_{l,q^s}$ and the strategy property $((C_1, C_1'), \ldots, (C_l, C_l'))$ the induced and valid constraint system $\mathbf{C} = \mathbf{C_B}$ (see below for the definition). Then, run the constraint solver on $\mathbf{C}$. If it returns no, then halt. Otherwise, let $\mathbf{C}'$ be the simple constraint system returned by the solver. (Recall that $\mathbf{C}'$ belongs to the sound and complete solution set and is chosen non-deterministically by the solver.)
3. Let ν be the complete solution associated with $\mathbf{C}'$. Check whether ν when applied to $\mathbf{B}$ yields a valid path in $\mathscr{G}_P$ from the initial state of P to a state q and q-strategy trees $\mathscr{T}_{i,q}$ satisfying (C_i, C_i') for every i. If so, output yes and $\mathbf{B}$ with ν applied, and otherwise return no (see below for more details). In case yes is returned, $\mathbf{B}$ with ν applied yields a concrete solution of the problem instance $(P, (C_1, C_1'), \ldots, (C_l, C_l'))$.

We emphasize that, for simplicity of presentation, SolveStrategy is formulated as a non-deterministic algorithm. Hence, the overall decision of SolveStrategy is yes if there exists at least one computation path where yes is returned. Otherwise, the overall decision is no (i.e., $(P, (C_1, C_1'), \ldots, (C_l, C_l')) \notin$ STRATEGY).

In the following, the three steps of SolveStrategy are further explained. The main result of [12, 14], is the following theorem:

Theorem 1. *SolveStrategy is a decision procedure for* STRATEGY.

Guess the Symbolic Branching Structure To describe the first step of SolveStrategy in more detail, we first define symbolic branching structures, which consist of symbolic paths and symbolic strategy trees. To define symbolic paths and strategy trees, we need to introduce symbolic states, transitions, and trees (see [14] for full details).

A *symbolic state* $q^s = ((\Pi_1, \ldots, \Pi_n), \mathscr{I}, \mathscr{S})$ is defined just as a concrete state (see Section 2.1) except that the substitution is omitted and the intruder knowledge $\mathscr{I}$ and the secure channel $\mathscr{S}$ may contain terms (with variables) instead of only messages. The *symbolic initial state* of a protocol $P = ((\Pi_1, \ldots, \Pi_n), \mathscr{I}_0)$ is $((\Pi_1, \ldots, \Pi_n), \mathscr{I}_0, \emptyset)$.

A *symbolic transition*, analogously to concrete transitions, is a transition between symbolic states and is of the form

$$((\Pi_1, \ldots, \Pi_n), \mathscr{I}, \mathscr{S}) \xrightarrow{\ell} ((\Pi'_1, \ldots, \Pi'_n), \mathscr{I}', \mathscr{S}') \tag{2}$$

with ℓ an appropriate label where again we distinguish between symbolic intruder, secure channel, and ε-transitions. Informally speaking, these transitions are of the following form (see [14] for details): For *symbolic intruder transitions* the label ℓ is of the form i, f, I where now f is not the message delivered by the intruder, as was the case for concrete intruder transitions, but a direct successor of the root r_i of Π_i. The intuition is that the principal rule $L \Rightarrow R$ the edge (r_i, f) is labeled with in Π_i is applied. The symbolic state $((\Pi_1, \ldots, \Pi_n), \mathscr{I}, \mathscr{S})$ is updated accordingly to $((\Pi'_1, \ldots, \Pi'_n), \mathscr{I}', \mathscr{S}')$. We call $L \Rightarrow R$ the *principal rule associated with the symbolic transition*. Similarly, the label of a *symbolic secure channel transition* is of the form i, f, L', sc where f is interpreted as before and L' is the term read from the secure channel. If $L \Rightarrow R$ is the principal rule associated with the transition, then $\mathscr{S}'$ is obtained by removing L' from $\mathscr{S}$ and adding R if R is a secure channel term. When constructing the constraint system, we will guarantee that L' unifies with L. Finally, the label of *symbolic ε-transitions* is of the form i, f with the obvious meaning.

A *symbolic q^s-tree* $\mathscr{T}^s_{q^s} = (V, E, r, \ell_V, \ell_E)$ is an unordered finite tree where the vertices are labeled with symbolic states, the root is labeled with q^s, and the edges are labeled with labels of symbolic transitions such that an edge (v, v') of the tree, more precisely, the labels of v and v' and the label of (v, v') correspond to symbolic transitions. We call the principal rule associated with such a symbolic transition *the principal rule associated with* (v, v'). Note that the symbolic transitions of different edges may be associated with the same principal rule. Now, since the same rule may occur at different positions in the tree, its variables may later be substituted differently. We therefore need a mechanism to consistently rename variables.

A *symbolic path* π^s of a protocol P is a symbolic q^s_0-tree where every vertex has at most one successor and q^s_0 is the symbolic initial state of P.

A *symbolic q^s-strategy tree* $\mathscr{T}^s_{q^s} = (V, E, r, \ell_V, \ell_E)$ is a symbolic q^s-tree which satisfies additional conditions. Among others, we require that in one node of this tree the intruder may only send a message to one principal Π_i; we show that this is w.l.o.g. Also, all ε-transitions applicable in one node are present. Symbolic strategy trees

are defined in such a way that for every symbolic state q^s the number of symbolic q^s-strategy trees is finite and all such trees can effectively be generated.

For a protocol P and strategy property $((C_1, C_1'), \ldots, (C_l, C_l'))$, a *symbolic branching structure* is of the form $\mathbf{B}^s = \pi^s, \mathscr{T}_1^s, \ldots, \mathscr{T}_l^s$ where π^s is a symbolic path of P and the $\mathscr{T}_i^s$ are symbolic q^s-strategy trees where q^s is the symbolic state the leaf of π^s is labeled with. Given a protocol and a strategy property, there are only a finite number of symbolic branching structures and these structures can be generated by an algorithm. In particular, there is a non-deterministic algorithm which can guess one symbolic branching structure $\mathbf{B}^s$ among all possible such structures.

Construct and Solve the Induced Constraint System In this step the constraint system $\mathbf{C} = \mathbf{C}_\mathbf{B}$ is derived from the symbolic branching structure $\mathbf{B} = \pi^s, \mathscr{T}_1^s, \ldots, \mathscr{T}_l^s$ (guessed in the first step of SolveStrategy) and the given strategy property $((C_1, C_1'), \ldots, (C_l, C_l'))$. This constraint system can be shown to be valid, and hence, by Fact 1, a constraint solver can be used to solve it.

We skip the details of this procedure (see [14] for full definition) and only informally explain how communication involving secure channels is handled. The basic idea is that messages intended for secure channel are written into the knowledge of the intruder's who may then deliver these messages. The problem is that while every message in the secure channel can only be read once, the intruder could try to deliver the same message several times. To prevent this, every such message when written into the intruder's knowledge is encrypted with a *new* key not known to the intruder and this key is also (and only) used in the principal rule which according to the symbolic branching structure is supposed to read the message. This guarantees that the intruder cannot abusively deliver the same message several times to unintended recipients or make use of these encrypted messages in other contexts.

Check the Induced Substitutions Let $\mathbf{B}^s = \pi^s, \mathscr{T}_1^s, \ldots, \mathscr{T}_l^s$ be the symbolic branching structure obtained in the first step of SolveStrategy and let $\mathbf{C}'$ be the simple constraint system returned by the constraint solver when applied to $\mathbf{C} = \mathbf{C}_{\mathbf{B}^s}$ in the second step of SolveStrategy. Let ν be the complete solution associated with $\mathbf{C}'$ (see Section 2.4). We emphasize that for our algorithm to work, it is important that ν replaces the variables in $\mathbf{C}'$ by *new* intruder atoms from $\mathscr{A}_I$ not occurring in $\mathbf{B}^s$.

Basically, we want to check that when applying ν to $\mathbf{B}^s$, which yields $\mathbf{B}^s\nu = \pi^s\nu, \mathscr{T}_1^s\nu, \ldots, \mathscr{T}_l^s\nu$, we obtain a solution of the problem instance $(P, (C_1, C_1'), \ldots, (C_l, C_l'))$. Hence, we need to check whether i) $\pi^s\nu$ corresponds to a path in $\mathscr{G}_P$ from the initial state of $\mathscr{G}_P$ to a state $q \in \mathscr{G}_P$ and ii) $\mathscr{T}_i^s\nu$ corresponds to a q-strategy tree for (C_i, C_i') for every i. However, since ν is a complete solution of $\mathbf{C}$, some of these conditions are satisfied by construction. In particular, $\pi^s\nu$ is guaranteed to be a path in $\mathscr{G}_P$ starting from the initial state. Also, the conditions 1.–3. of strategy trees (Definition 1) do not need to be checked and we know that $\mathscr{T}_i^s\nu$ satisfies $(C_i, \emptyset)$. Hence, SolveStrategy only needs to make sure that 4. and 5. of Definition 1 are satisfied for every $\mathscr{T}_i^s\nu$ and that $\mathscr{T}_i^s\nu$ fulfills $(\emptyset, C_i')$. Using that the derivation problem is decidable in polynomial time [8] (given a message m and a finite set of

messages $\mathcal{I}$, decide whether $m \in d(\mathcal{I})$), all of these remaining conditions can easily be checked (see [14] for details).

3 Implementation

We have implemented the algorithm proposed in [12, 14], along with several optimizations (see [15] for our implementation). The implementation is written in Prolog and consists of about 2500 lines of code. In our implementation we used the constraint solver by Corin and Etalle [10] (which is based on the constraint solver by Millen and Shmatikov [16]). This constraint solver, although not the most efficient one, is written also in Prolog and therefore could easily be integrate with our implementation.

3.1 Optimizations

Our first, naive implementation of the constraint solving algorithm presented in [12, 14] was hardly usable, due to severe performance problems. As discussed in the introduction, this was expected: the algorithm needs to test exponentially many strategy tries, each of which inducing a constraint system of significant size to be solved by the constraint solver (note that solving a constraint system is an NP-hard problem).

Therefore we had to introduce several optimizations in order to make the algorithm usable. In the following, we describe the main optimizations.

Constructing strategy trees: To describe this optimization, we first need to introduce what we call standard strategy trees and standard symbolic branching structures. A strategy tree (a symbolic branching structure) is *standard* if the following condition is satisfied: a (symbolic) intruder transition can originate from a given (symbolic) state only if there is no epsilon or secure-channel transition originating from this state.

It is easy to show that it suffices to consider only standard strategy trees (symbolic branching structures): If there exists a strategy tree T that satisfies a given property (C, C'), there also exists a standard strategy tree T' that satisfies (C, C'). In fact, we can obtain T' from T by simply removing all the intruder transitions originating from states which have also some outgoing epsilon or secure-channel transitions. We can note that in this way we obtain a valid strategy tree (assuming that T is a valid strategy tree), i.e. T' satisfies all the conditions of Definition 1, if T does. Moreover, every leaf of T' is also a leaf of T. This immediately implies that whenever T satisfies (C, C'), then T' satisfies (C, C') as well.

This observation allows us to make the following optimization: when a symbolic branching structure is guessed (see page 8), we consider only standard symbolic branching structures. This significantly reduces the number of branching structures to be tested, as well as their size.

Secure channel encoding: The algorithm, as described in [12, 14], handles secure channels in the following way: When a constraint system is built for a given strategy tree, communication through secure channels is modelled by introducing a fresh key $k_{a,b}$, for every secure channel (a, b). Then, a message sent over such a channel is encrypted using $k_{a,b}$ and added to the intruder knowledge and, for each participant rule $R \Rightarrow S$ that reads from this channel (i.e. with R of the form $\mathsf{sc}(a, b, t)$), the term R is encoded as $\{t\}^s_{k_{a,b}}$.

Although this is a sound encoding, it introduces some overhead to the constraint solving algorithm: The constraint solver, when deriving $\{t\}^s_{k_{a,b}}$, tries two possibilities: First, it tries to unify $\{t\}^s_{k_{a,b}}$ with some other message m (which is possible only if $m = \{m'\}^s_{k_{a,b}}$ and m' unifies with t). Second, it tries to derive t and $k_{a,b}$ separately. As one can show, the constraint solver never succeeds in the latter case (as the key $k_{a,b}$ is never revealed) and therefore this step can safely be omitted.

In our implementation, we handle secure channel in a similar, but slightly different way: instead of using the message $\{t\}^s_{k_{a,b}}$, as described above, we use $f(t, k_{a,b})$, where f is a free function symbol (i.e. there is no decomposing rule for f in the constraint solver). Owing to this modification, the constraint solver avoids spending time trying to derive t and $k_{a,b}$ (the only way to derive $f(t, k_{a,b})$ is now to unify it with some $m = f(m', k_{a,b})$ such that m' unifies with t).

Treatment of secure channels: This simplification is based on the observation that typically the following is true for every channel from a to b, where both a and b are honest protocol participants: whenever some message has been sent on this channel (technically, in a given state a message of the form $\mathsf{sc}(a, b, m)$ is in $\mathscr{S}$) and b is ready to read a message from this channel (technically, there is a principal rule $R \Rightarrow S$ of b in this state, with R of the form $\mathsf{sc}(a, b, t)$), then this message can be delivered to b (technically, t matches with m).

This observation allows us to determine, already in a symbolic strategy branching structure, which secure channel transition must be present. Owing to this, after the second step of SolveStrategy, when the algorithm checks the induced substitution as described on page 9, not only condition 1.–3. of strategy trees (Definition 1) do not need to be checked, but also condition 4. is satisfied by the construction of the branching structure and does not need to be checked. Therefore, in this step, only condition 5. of Definition 1 has to be checked.

This modification allows us to eliminate some strategy trees before the constraint-solving algorithm is executed. We note that while the modification does not preserve soundness of our procedure in general, it is sound for the cases typically encountered in applications; and this can be checked easily.

4 Experiments

We applied our implementation to two protocols. The first one is a simple, toy protocol given in [14]. We used this protocol to test and debug our implementation. The second one is the ASW protocol, a contract-signing protocol proposed in [4].

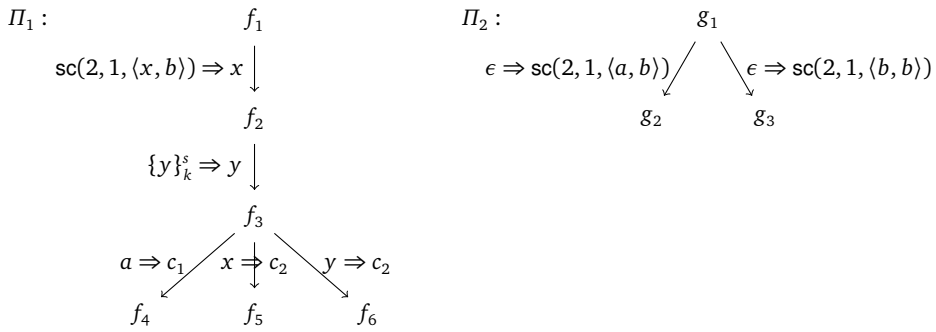

Fig. 1. Protocol $P_{\mathsf{ex}} = (\{\Pi_1, \Pi_2\}, \mathscr{I}_0)$ with $\mathscr{I}_0 = \{\{a\}^s_k, \{b\}^s_k\}$

4.1 The Simple Protocol

This protocol, depicted in Figure 1, consists of two principals Π_1 and Π_2 and the initial knowledge $\mathscr{I}_0 = \{\{a\}^s_k, \{b\}^s_k\}$ of the intruder. Informally speaking, Π_2 can, without waiting for input from the secure channel or the intruder, decide whether to write $\langle a, b\rangle$ or $\langle b, b\rangle$ into the secure channel from Π_2 to Π_1. While the intruder can read the message written into this channel, he cannot modify or delay this message. Also, he cannot insert his own message into this channel as he does not have the principal address 2 in his intruder knowledge, and hence, cannot generate messages of the form $\mathsf{sc}(2, \cdot, t)$. Consequently, such messages must come from Π_2. Principal Π_1 first waits for a message of the form $\langle x, b\rangle$ in the secure channel from Π_2 to Π_1. In case Π_2 wrote, say, $\langle a, b\rangle$ into this channel, x is substituted by a, and this message is written into the network, and hence, given to the intruder. Next, Π_1 waits for input of the form $\{y\}^s_k$. This is not a secure channel term, and thus, comes from the intruder. In case the intruder sends $\{b\}^s_k$, say, then y is substituted by b. Finally, Π_1 waits for input of the form a (in the edges from f_3 to f_4 and f_3 to f_5) or b (in the edge from f_3 to f_6). Recall that x was substituted by a and y by b. If the intruder sends b, say, then Π_2 takes the edge from f_3 to f_6 and outputs c_2 into the network. If the intruder had sent a, Π_1 could have chosen between the first two edges.

We tested this protocol for several natural strategy properties. In each case, the run-time of our tool was very short (a fraction of a second).

4.2 ASW

Before describing the results of our experiments, we briefly recall the ASW protocol [4].

Overview of the Protocol: Our informal description of the ASW protocol follows [18] (see this work or [4] for more details). For ease in notation, we will write $\mathsf{sig}[m, k]$ instead of $\langle m, \mathsf{sig}_k(m)\rangle$.

The ASW protocol enables two principals O (originator) and R (responder) to obtain each other's commitment on a previously agreed contractual text, say text, with the help of a trusted third party T, which, however, is only invoked in case of problems. In other words, the ASW protocol is an optimistic two-party contract-signing protocol.

There are two kinds of valid contracts: the standard contract,

$$\langle \mathsf{sig}[m_O, k_O], N_O, \mathsf{sig}[m_R, k_R], N_R \rangle \quad ,$$

and the replacement contract, $\mathsf{sig}[\langle \mathsf{sig}[m_O, k_O], \mathsf{sig}[m_R, k_R] \rangle, k_T]$, where

$$m_O = \langle k_O, k_R, k_T, \mathsf{text}, \mathsf{hash}(N_O) \rangle \quad ,$$
$$m_R = \langle \mathsf{sig}[m_O, k_O], \mathsf{hash}(N_R) \rangle \quad ,$$

and k_O, k_R, and k_T are the public keys of O, R, and T, respectively, and N_O and N_R represent nonces generated by O and R, respectively. Note that a signed contractual text ($\mathsf{sig}[\mathsf{text}, k_O]$ or $\mathsf{sig}[\mathsf{text}, k_R]$) is not considered a valid contract.

The ASW protocol consists of four subprotocols: the exchange, abort, and two resolve protocols. However, we can describe every principal—O, R, and T—in terms of a single tree as introduced in Section 2.1.

The basic idea of the exchange protocol is that O first indicates his/her interest to sign the contract. To this end, O hashes a nonce N_O and signs it together with text and the keys of the principals involved. The resulting message is the message $\mathsf{sig}[m_O, k_O]$ from above. By sending it to R, the originator O commits to the contract. Then, similarly, R indicates his/her interest to sign the contract by hashing a nonce N_R and signing it together with what he/she received from O and the keys of the involved principals. This is the message $\mathsf{sig}[m_R, k_R]$ from above. By sending it to O, the responder R commits to the contract. Finally, first O and then R reveal N_O and N_R, respectively. This is why a standard contract is only valid if N_O and N_R are included.

If, after O has sent the first message, R does not respond, O may contact T to abort. Also, if after O has sent the second message, the nonce N_O, and R does not respond, then O may contact T to resolve. Similarly, R may contact T to resolve the protocol after having sent the message $\mathsf{sig}[m_R, k_R]$. However, R may not abort the protocol. In case the protocol is successfully resolved, the replacement contract $\mathsf{sig}[\langle \mathsf{sig}[m_O, k_O], \mathsf{sig}[m_R, k_R] \rangle, k_T]$ is issued. While this version of the contract only contains the message indicating O's and R's intention to sign the contract (and neither N_O nor N_R), the signature of T validates the contract. For T to respond to incoming resolve or abort requests appropriately it needs to store the requests in some kind of database. For example, if O has sent an abort request and afterwards R sends a resolve request T has to answer by an abort token to act consistently.

As we will see in the next subsection the model of the originator O of the ASW protocol as a rule tree is straightforward. To model the TTP T as a rule tree one has to encode the database which is needed to distinguish between different possible interleavings of resolve and abort requests. This can be done by unwinding these possible interleavings into a rule tree.

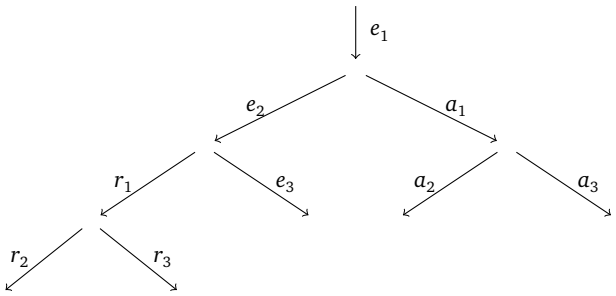

Fig. 2. A model of the Originator O in the ASW protocol

The Principal O: The principal O is defined by the tree Π_O depicted in Figure 2 where the edge labels for the principal rules defined below. Rules $e1$, $e2$, and $e3$ belong to the exchange protocol, rules $a1$, $a2$, and $a3$ belong to the abort protocol, and rules $r1$, $r2$, and $r3$ belong to the resolve protocol of O.

Exchange protocol. The actions performed in the exchange protocol have informally been discussed above.

Abort protocol. If, after the first step of the exchange protocol, O does not get an answer back from R, the principal O may start the abort protocol, i.e., send an abort request via a secure channel to T (rule $a1$). Then, T will either confirm the abort of the protocol by returning an abort token—in this case O will continue with rule $a3$—or send a replacement contract—in this case O will continue with rule $a2$. (The trusted third party T sends a replacement contract if R previously contacted T to resolve the protocol run.)

Resolve protocol. If after rule $e2$, i. e., after sending N_O, the principal O does not get an answer back from R, then O can start the resolve protocol by sending a resolve request to T via the secure channel (rule $r1$). After that, depending on the answer returned from T (which again will return an abort token or a replacement contract), one of the rules $r2$ or $r3$ is performed.

We now present the principal rules for O where the numbering corresponds to the one in Figure 2. In some of these rules we use extra constants which indicate certain events. We therefore call these constants *signal constants*. If, for example, principal O performs rule e_3, i.e., it gets the signature of the responder on the contract, it sends the signal constant OHasValidContract on the network, indicating that O now has a valid contract. Then, the property that at the end of a protocol execution the originator O does not have a valid contract is formalized by requiring that the intruder cannot derive the constant OHasValidContract at the end of a protocol execution. How security properties are modeled in our framework is described below.

(e_1) $\varepsilon \Rightarrow me_1$ where

$$me_1 = \mathsf{sig}[me_2, k_O] \quad \text{and} \quad me_2 = \langle k_O, k_R, k_T, \mathsf{text}, \mathsf{hash}(N_O)\rangle\,.$$

(e_2) $\mathsf{sig}[me_3, k_R] \Rightarrow N_O$ where $me_3 = \langle me_1, \mathsf{hash}(x)\rangle\,.$

(e_3) $x \Rightarrow \mathsf{OHasValidContract}$.
(a_1) $\varepsilon \Rightarrow \mathsf{sc}(O,T,ma_1)$ where $ma_1 = \mathsf{sig}[\langle \mathsf{aborted}, me_1 \rangle, k_O]$.
(a_2) $\mathsf{sc}(T,O,ma_2) \Rightarrow \mathsf{OHasValidContract}$ where

$$ma_2 = \mathsf{sig}[\langle me_1, me_4 \rangle, k_T] \quad \text{and} \quad me_4 = \mathsf{sig}[\langle me_1, z_1 \rangle, k_R].$$

(a_3) $\mathsf{sc}(T,O,\mathsf{sig}[\langle \mathsf{aborted}, ma_1 \rangle, k_T]) \Rightarrow \mathsf{OHasAbortToken}$.
(r_1) $\varepsilon \Rightarrow \mathsf{sc}(O,T,\langle me_1, \mathsf{sig}[me_3, k_R] \rangle)$.
(r_2) $\mathsf{sc}(T,O,\mathsf{sig}[\langle \mathsf{aborted}, ma_1 \rangle, k_T]) \Rightarrow \mathsf{OHasAbortToken}$.
(r_3) $\mathsf{sc}(T,O,mr_1) \Rightarrow \mathsf{OHasValidContract}$ where

$$mr_1 = \mathsf{sig}[\langle me_1, mr_2 \rangle, k_T] \quad \text{and} \quad mr_2 = \mathsf{sig}[\langle me_1, z_2 \rangle, k_R].$$

Experiments: We have applied our tool to the ASW protocol considering the following scenarios:

1. *Honest Alice:* In this scenario we assume the originator (Alice), as well as the trusted third party, to be honest. Bob is assumed to be dishonest and is subsumed by the intruder.
2. *Honest optimistic Alice:* In this scenario, as before, we assume the originator (Alice) and the trusted third party to be honest, with dishonest Bob subsumed by the intruder. This time, however, Alice is assumed to be *optimistic*, i.e., she is willing to wait for messages of other parties. Technically, Alice only contacts the TTP if the dishonest responder (the intruder) allows her to do so. In other words, the dishonest responder can buy himself as much time as he needs, before the honest signer contacts the TTP. This is a reasonable assumption, also made in [7].
3. *Honest Bob:* In this scenario we assume the responder (Bob) as well as the trusted third party to be honest. The originator (Alice) is assumed to be dishonest and is subsumed by the intruder.

We verified these scenarios against the well known *balance* property which requires that in no state of a protocol execution the intruder (a dishonest party) has both (a) a strategy to obtain a valid contract and (b) a strategy to prevent the second (honest) participant from obtaining a valid contract. In the case of honest originator (Alice), this property is captured by the strategy property

$$\phi_O = (((\{\mathsf{RHasValidContract}\}, \emptyset), (\emptyset, \{\mathsf{OHasValidContract}\})).$$

More precisely, to guarantee the balance property, we require that the above strategy property is *not* satisfied in the protocol. Similarly, in the case of honest responder, balance is captured by the strategy property

$$\phi_R = (((\{\mathsf{OHasValidContract}\}, \emptyset), (\emptyset, \{\mathsf{RHasValidContract}\})).$$

Note that the ASW protocol is supposed to guarantee the balance property in the first and the third scenario above. Only if Alice is honest and optimistic (the second

scenario), the balance property is known to be violated (and hence, one can expect that the corresponding strategy property ϕ_O is satisfied).

When we applied our tool to the three scenarios described above, the program, unfortunately, did not terminate (in reasonable time). The reason for this was that the constraint systems generated by the algorithm turned out to be too big for the constrained solver we used. Therefore, we applied some sound simplifications to our modeling:

1. We removed signatures generated by dishonest parties (the honest parties after this simplification, instead of messages signed by the dishonest party, expect unsigned messages). It is easy to see that this is a sound modification: if there exists an attack (a strategy tree) in the original protocol, then there also exists an attack in the modified variant (the same strategy tree but with stripped off signatures of dishonest participants). Moreover, we can note that this modification does not introduce any new attacks, as the intruder is able to generate signatures of dishonest parties for any message he generates.
2. Whenever, in the original protocol, a message $\mathsf{sig}[m,k] = \langle m, \mathsf{sig}_k(m)\rangle$ is sent or received by an honest party, where k is a signing key of an honest party and m is some message known to the intruder, then we send/receive $\mathsf{sig}_k(m)$, instead. Note that, by this modification, we do not restrict the intruder in any way, as m is known to him. Therefore, whenever some message can be sent/delivered in the original protocol, the corresponding message (where $\mathsf{sig}[m,k]$ is replaced by $\mathsf{sig}_k(m)$) can be sent/delivered in the modified version. This shows that the modification is sound.

After these (sound) modifications, we were able to successfully apply our tool to all the mentioned variants of the protocol. The experiments were carried on a PC with 2,4 Ghz Intel CoreTM 2 Duo E6700 processor and 2GB RAM. The results were obtained in less than 2 minutes in each case (in fact, for honest Alice and honest optimistic Alice, the running times were about 10 seconds). The results were as expected: in the second scenario an attack was found; in the remaining two, no attack was detected.

As mentioned, the efficiency problem for the original ASW protocol does not stem from a too big number of strategy trees to be checked, but rather from the size of the generated constraint systems which turned out to be too big for the constraint solver we used. This solver, however, is not the most efficient one. Therefore we believe that using a more efficient constraint solver could improve the overall performance of our tool in such a significant way that the above mentioned modification of ASW would not be necessary. But we leave this as future work.

5 Conclusion

We have implemented the constraint-solving algorithm for checking game-theoretic security properties proposed in [12, 14] along with several optimizations. Our implementation is promising in that it shows that in principle automatic analysis of

game-theoretic security properties is possible for non-trivial contract-signing protocols. However, further efforts are necessary to optimize our implementation. Also, new cryptographic primitives, such as private contract signatures, should be considered to be able to broaden the scope of our implementation. Finally, while our implementation is based on a hardly optimized constraint solver, it seems fruitful to base our implementation on a more efficient constraint solver in order to be able to handle more complex constraint systems.

References

1. R.M. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2002.
2. A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In E. Brinksma and K. G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 349–353. Springer, 2002.
3. A. Armando, D.A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P.H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In K. Etessami and S.K. Rajamani, editors, *Computer Aided Verification, 17th International Conference (CAV 2005)*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer-Verlag, 2005.
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. IEEE Computer Society, 1998.
5. D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In E. Snekkenes and D. Gollmann, editors, *Proceedings of the 8th European Symposium on Research in Computer Security (ESORICS 2003)*, volume 2808 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2003.
6. R. Chadha, M.I. Kanovich, and A.Scedrov. Inductive methods and contract-signing protocols. In P. Samarati, editor, *8-th ACM Conference on Computer and Communications Security (CCS 2001)*, pages 176–185. ACM Press, 2001.
7. R. Chadha, J.C. Mitchell, A. Scedrov, and V. Shmatikov:. Contract Signing, Optimism, and Advantage. In R.M. Amadio and D. Lugiez, editors, *CONCUR 2003 - Concurrency Theory, 14th International Conference*, volume 2761 of *Lecture Notes in Computer Science*, pages 361–377. Springer, 2003.
8. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *Proceedings of the Eighteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 261–270. IEEE Computer Society, 2003.
9. Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proceedings of the 16th IEEE Conference on Automated Software Engineering (ASE 2001)*, pages 373–376. IEEE CS Press, 2001.
10. R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In M.V. Hermenegildo and G. Puebla, editors, *Proceedings of the 9th International Symposium on Static Analysis (SAS 2002)*, volume 2477 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2002.

11. J.A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology – CRYPTO'99, 19th Annual International Cryptology Conference*, volume 1666 of *Lecture Notes in Computer Science*, pages 449–466. Springer-Verlag, 1999.
12. D. Kähler and R. Küsters. Constraint Solving for Contract-Signing Protocols. In M. Abadi and L. de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR 2005)*, volume 3653 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2005.
13. D. Kähler, R. Küsters, and Th. Wilke. Deciding Properties of Contract-Signing Protocols. In V. Diekert and B. Durand, editors, *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS 2005)*, volume 3404 of *Lecture Notes in Computer Science*, pages 158–169. Springer-Verlag, 2005.
14. D. Kähler, R. Küsters, and Th. Wilke. Deciding Strategy Properties of Contract-Signing Protocols. *ACM Transactions on Computational Logic (TOCL)*, 11(3), 2010. Printed version to appear.
15. Ralf Küsters, Thomas Schmidt, and Tomasz Truderung. CSVer: a Protocol Verifier for Contract-Signing Protocols, 2011. Available at http://infsec.uni-trier.de/tools.html.
16. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 166–175. ACM Press, 2001.
17. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *Theoretical Computer Science*, 299(1–3):451–475, 2003.
18. V. Shmatikov and J.C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science (TCS), special issue on Theoretical Foundations of Security Analysis and Design*, 283(2):419–450, 2002.