Constraint Solving for Contract-Signing Protocols

Detlef Kähler and Ralf Küsters

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
{kaehler,kuesters}@ti.informatik.uni-kiel.de

Abstract. Research on the automatic analysis of cryptographic protocols has so far mainly concentrated on reachability properties, such as secrecy and authentication. Only recently it was shown that certain game-theoretic security properties, such as balance for contract-signing protocols, are decidable in a Dolev-Yao style model with a bounded number of sessions but unbounded message size. However, this result does not provide a practical algorithm as it merely bounds the size of attacks. In this paper, we prove that game-theoretic security properties can be decided based on standard constraint solving procedures. In the past, these procedures have successfully been employed in implementations and tools for reachability properties. Our results thus pave the way for extending these tools and implementations to deal with game-theoretic security properties.

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., [14, 1]). Based on this result, many fully automatic tools (see, e.g., [2, 7, 13]) have been developed and successfully been applied to find flaws in published protocols, where many of these tools employ so-called *constraint solving procedures* (see, e.g., [13, 7, 4]). 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., [9, 3]), for instance abuse-freeness [9] and balance [5], 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.

Only recently [11], the central decidability result mentioned above was extended to such game-theoretic security properties, including, for instance, balance. However, similar to the result by Rusinowitch and Turuani [14] for reachability properties, the decision algorithm presented in [11] 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. Clearly, just as in the case of reachability properties, this is completely impractical. For reachability properties, one has therefore developed the mentioned constraint solving procedures to obtain practical decision algorithms.

The main contribution of the present work is a *constraint-based* decision algorithm for the game-theoretic security properties of the kind considered in [11]. The main feature of our algorithm is that it can be built on top of *standard constraint solving procedures* (see, e.g., [13, 7, 4] 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, our algorithm paves the way for extending existing implementations and tools for reachability properties to deal with game-theoretic security properties.

In a nutshell, our constraint-based algorithm works as follows: Given a protocol along with the considered game-theoretic security property, first the algorithm guesses what we call a symbolic branching structure. This structure represents a potential attack on the protocol and corresponds to the interleavings, which are, however, linear structures, guessed for reachability properties. In the second step of the algorithm, the symbolic branching structure is turned into a constraint system. This step requires some care due to the branching issue and write-protected channels considered in our model (also called secure channels here), i.e., channels that are not under the control of the intruder. Then, a standard constraint solving procedure is used to compute a finite sound and complete set of so-called simple constraint systems. A simple constraint system in such a set represents a (possibly infinite) set of solutions of the original constraint system and the sound and complete set of these simple constraint systems represents the set of *all* solutions of the original constraint system. Finally, it is checked whether (at least) one of the computed simple constraint systems in the sound and complete set passes certain additional tests.

There are some crucial differences of our constraint-based algorithm to algorithms for reachability properties: First, as mentioned, instead of symbolic branching structures, for reachability properties only interleavings, i.e., linear structures, need to be guessed. Turning these interleavings into constraint systems is immediate due to the absence of the branching issue and the absence of secure channels. Second, and more importantly, for reachability properties it suffices if the constraint solving procedure only returns one simple constraint system, rather than a sound and complete set. Third, the final step of our constraint-based algorithm—performing additional tests on the simple constraint system—is not required for reachability properties.

We emphasize that even though for reachability properties it suffices if the constraint solving procedure returns only one simple constraint system, standard constraint solving procedures are typically capable of computing sound and complete sets of simple constraint systems. Any such procedure can be used by our constraint-based algorithm as a black-box for solving constraint systems. This makes it possible to extend existing implementations and tools for reachability properties to deal with game-theoretic properties since the core of the algorithms—solving constraint systems—remains the same, provided that the considered cryptographic primitives can be dealt with by the constraint solving procedure (see Section 4).

The protocol and intruder model that we use is basically the one proposed in [11], which in turn is the "bounded session" version of a model proposed in [5]. We slightly modify the model of [11]—without changing its expressivity and accuracy—in order to simplify our constraint-based algorithm (see Section 2 and 3).

Further related work. Contract-signing and related protocols have been analyzed both manually [5], based on a relatively detailed model (as mentioned before, our model is a "bounded session" version of this model), and using finite-state model checking (see, e.g., [15, 12]), based on a coarser finite-state model. Drielsma and Mödersheim [8] were the first to apply an automatic tool based on constraint solving to the contract-signing protocol by Asokan, Shoup, and Waidner [3]. Their analysis is, however, restricted to reachability properties since game-theoretic properties cannot be handled by their tool. The results shown in the present work pave the way for extending such tools in order to be able to analyze game-theoretic properties.

Structure of this paper. We first introduce our protocol and intruder model (Section 2) as well as intruder strategies and game-theoretic properties (Section 3). Section 4 provides the necessary background on constraint solving. In Section 5, we present our constraint-based decision algorithm along with an example and state our main result—soundness, completeness, and termination of the algorithm. We conclude in Section 6. Full definitions and proofs can be found in our technical report [10].

2 The Protocol and Intruder Model

The protocol and intruder model that we use basically coincides with the model first introduced in [11], which in turn is the "bounded session" version of the model proposed in [5]. We only slightly modify the model in [11] in that we impose a restriction on principals which is necessary for principals to perform feasible computations.

In our model, 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., [14]) inputs to principals are always provided by the intruder, in our model inputs can also come from the secure channel. Secure channels are typically used by principals in contract-signing protocols to communicate with a trusted third party. The important point is that these channels 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). However, just as in [5], the intruder can read the messages written onto the secure channel. We note that our results also hold in case of read-protected secure channels. Another difference to standard Dolev-Yao models is that, in order to be able to formulate game-theoretic properties, we explicitly describe the behavior of a protocol as an infinite-state transition graph which comprises all runs of a protocol.

We now describe the model in more detail by defining terms and messages, the intruder, principals and protocols, and the transition graph.

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 the smallest set satisfying the following conditions: $\mathcal{I} \subseteq d(\mathcal{I})$; if $m, m' \in d(\mathcal{I})$, then $\langle m, m' \rangle \in d(\mathcal{I})$; if $\langle m, m' \rangle \in d(\mathcal{I})$, then $m \in d(\mathcal{I})$ and $m' \in d(\mathcal{I})$; if $m, m' \in d(\mathcal{I})$, then $\{m\}^s_{m'} \in d(\mathcal{I})$; if $\{m\}^s_{m'} \in d(\mathcal{I})$ and $m' \in d(\mathcal{I})$, then $m \in d(\mathcal{I})$; if $m \in d(\mathcal{I})$ and $k \in d(\mathcal{I}) \cap \mathcal{K}$, then $\{m\}^a_k \in d(\mathcal{I})$; if $\{m\}^a_k \in d(\mathcal{I})$ and $k^{-1} \in d(\mathcal{I})$, then $m \in d(\mathcal{I})$; if $m \in d(\mathcal{I})$, then $\mathsf{hash}(m) \in d(\mathcal{I})$; if $m \in d(\mathcal{I})$ and $k^{-1} \in d(\mathcal{I}) \cap \mathcal{K}$, then $\mathsf{sig}_k(m) \in d(\mathcal{I})$ (the signature contains the public key but can only be generated if the corresponding private key is known); 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*); $\mathcal{A}_I \subseteq d(\mathcal{I})$ (*generating fresh constants*).

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 our model, all (strongly) dishonest parties are subsumed in the intruder. Weakly dishonest parties can be modeled as principals whose specification deviates from the specification of the protocol.

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 tuple consisting of a rule tree $\Pi = (V, E, r, \ell)$ and a finite set of plain messages, the *initial knowledge of the principal*. Similar to models for reachability properties, we require that every variable occurring on the right-hand side of a principal rule $\ell(v, v')$ in Π also occurs on the left-hand side of $\ell(v, v')$ or on the left-hand side of a principal rule on the path from r to v. In addition, and unlike [11], we require a condition necessary for the principal to perform a feasible computation: The decryption

and signature verification operations performed when receiving a message can actually be carried out, i.e., terms in key positions (t' in $\{t\}^s_{t'}$, k^{-1} in $\{t\}^a_k$, and k in $\mathsf{sig}_k(t)$) on the left-hand side of principal rules can be derived from the set consisting of the left-hand side of the current principal rule, the left-hand sides of preceeding rules, and the initial knowledge of the principal. Obviously, the above condition is satisfied for all realistic principals. Moreover, it allows to simplify the constraint-based algorithm (Section 5).

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* $P = ((\Pi_1, \ldots, \Pi_n), \mathcal{I})$ consists of a finite sequence of principals Π_i and a finite set $\mathcal{I}$ 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 $\mathcal{A}_I$, do not occur in P.

As an example protocol, let us consider P_{ex} as depicted in Figure 1. This protocol consists of two principals Π_1 and Π_2 and the initial knowledge $\mathcal{I}_0 = \{\{a\}^s_k, \{b\}^s_k\}$ 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 note that this protocol is not meant to perform a useful task. It is rather used to illustrate different aspects of our constraint-based algorithm ([11] contains a formal specification of the contract-signing protocol by Asokan, Shoup, and Waidner [3] in our model).

2.1 Transition Graph Induced by a Protocol

A transition graph $\mathcal{G}_P$ 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 $((\Pi_1, \ldots, \Pi_n), \sigma, \mathcal{I}, \mathcal{S})$ where σ is a ground substitution, for each i, Π_i is a rule tree such that $\Pi_i\sigma$ is a principal, $\mathcal{I}$ is a finite set of messages, the *intruder knowledge*, and $\mathcal{S}$ 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

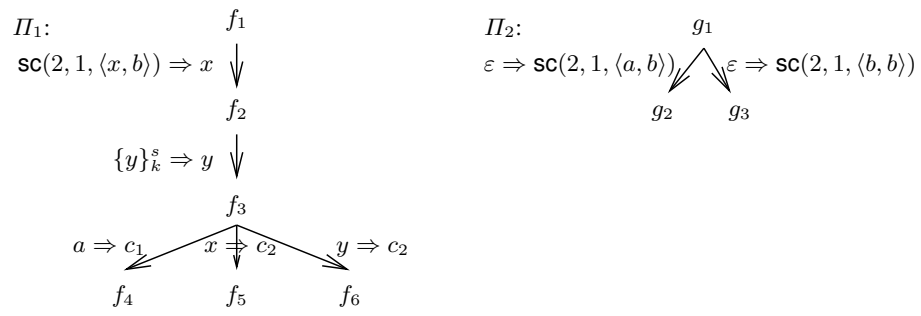

Fig. 1. Protocol $P_{ex} = (\{\Pi_1, \Pi_2\}, \mathcal{I}_0)$ with $\mathcal{I}_0 = \{\{a\}^s_k, \{b\}^s_k\}$, initial knowledge $\{1, a, b, k, c_1, c_2\}$ of Π_1 and initial knowledge $\{2, a, b\}$ of Π_2.

be derived from $\mathcal{I}$, the secure channels hold the messages in $\mathcal{S}$, and for each i, Π_i is the "remaining protocol" to be carried out by principal i. This also explains why $\mathcal{S}$ is a multi-set: messages sent several times should be delivered several times. Given a protocol $P = ((\Pi_1, \ldots, \Pi_n), \mathcal{I})$ the *initial state of P* is $((\Pi_1, \ldots, \Pi_n), \sigma, \mathcal{I}, \emptyset)$ where σ is the substitution with empty domain.

We have 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. We define under which circumstances there is a transition

$$((\Pi_1, \ldots, \Pi_n), \sigma, \mathcal{I}, \mathcal{S}) \xrightarrow{\tau} ((\Pi'_1, \ldots, \Pi'_n), \sigma', \mathcal{I}', \mathcal{S}') \tag{1}$$

with τ an appropriate label.

1. *Intruder transitions:* The transition (1) with label i, m, I exists if there exists $v \in V_i$ with $(r_i, v) \in E_i$ and $\ell_i(r_i, v) = L \Rightarrow R$, and a substitution σ'' of the variables in $L\sigma$ such that (a) $m \in d(\mathcal{I})$, (b) $\sigma' = \sigma \cup \sigma''$, (c) $L\sigma' = m$, (d) $\Pi'_j = \Pi_j$ for every $j \neq i$, $\Pi'_i = \Pi_i{\downarrow}v$, (e) $\mathcal{I}' = \mathcal{I} \cup \{R\sigma'\}$ if $R \neq \mathsf{sc}(\cdot, \cdot, \cdot)$, and $\mathcal{I}' = \mathcal{I} \cup \{t\sigma'\}$ if $R = \mathsf{sc}(\cdot, \cdot, t)$ for some t, (f) $\mathcal{S}' = \mathcal{S}$ if $R \neq \mathsf{sc}(\cdot, \cdot, \cdot)$, and $\mathcal{S}' = \mathcal{S} \cup \{R\sigma'\}$ otherwise. This transition models that principal i reads the message m from the intruder (i.e., the public network).
2. *Secure channel transitions:* The transition (1) with label i, m, sc exists if there exists $v \in V_i$ with $(r_i, v) \in E_i$ and $\ell_i(r_i, v) = L \Rightarrow R$, and a substitution σ'' of the variables in $L\sigma$ such that $m \in \mathcal{S}$, (b)–(e) from 1., and $\mathcal{S}' = \mathcal{S} \setminus \{m\}$ if $R \neq \mathsf{sc}(\cdot, \cdot, \cdot)$, and $\mathcal{S}' = (\mathcal{S} \setminus \{m\}) \cup \{R\sigma'\}$ otherwise. This transition models that principal i reads message m from the secure channel.
3. *ε-transitions:* The transition (1) with label i exists if there exists $v \in V_i$ with $(r_i, v) \in E_i$ and $\ell_i(r_i, v) = \varepsilon \Rightarrow R$ such that $\sigma' = \sigma$ and (d), (e), (f) from above. This transition models that i performs a step where neither a message is read from the intruder nor from the 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 $\mathcal{G}_P$.

We note that $\mathcal{G}_P$ is a DAG since by performing a transition, the size of the first component of a state decreases. While the graph may be infinite branching, the maximal length of a path in this graph is bounded by the total number of edges in the principals Π_i of P.

3 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).

More concretely, let us consider the protocol P_{ex} depicted in Figure 1. We want to know if the intruder has a strategy to get to a state where he can derive atom c_2 but not atom c_1 (no matter what the principals Π_1 and Π_2 do). Such a strategy of the intruder has to deal with both decisions principal Π_2 may make in the first step because the intruder cannot control which edge is taken by Π_2. It turns out that regardless of which message is sent by principal Π_2 in its first step, the following simple strategy allows the intruder to achieve his goal: The intruder can send $\{b\}_k^s$ to principal Π_1 in the second step of Π_1 and in the last step of Π_1, the intruder sends b to principal Π_1. This guarantees that in the last step of Π_1, the left-most edge is never taken, and thus, c_1 is not returned, but at least one of the other two edges can be taken, which in any case yields c_2. Formally, such strategies are defined as trees. In our example, the strategy tree corresponding to the strategy informally explained above is depicted in Figure 2.

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.)*

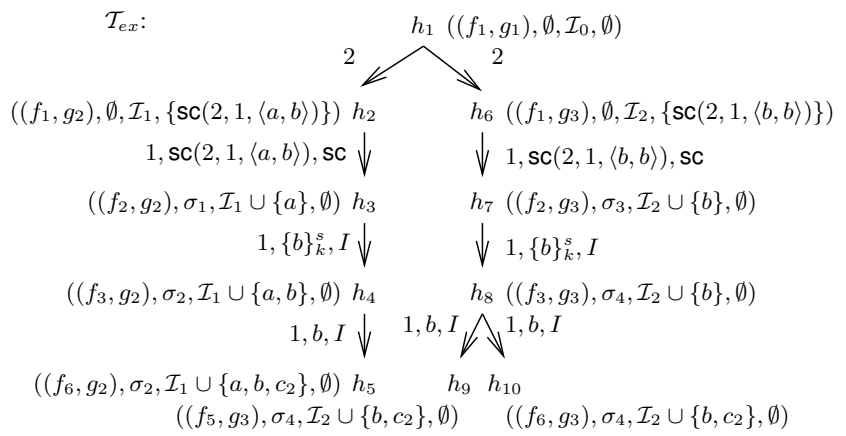

Fig. 2. Strategy tree $\mathcal{T}_{ex}$ for P_{ex} with $\mathcal{I}_1 = \mathcal{I}_0 \cup \{\langle a, b\rangle\}$, $\mathcal{I}_2 = \mathcal{I}_0 \cup \{\langle b, b\rangle\}$, $\sigma_1 = \{x \mapsto a\}$, $\sigma_2 = \sigma_1 \cup \{y \mapsto b\}$, $\sigma_3 = \{x \mapsto b\}$, and $\sigma_4 = \sigma_3 \cup \{y \mapsto b\}$. Also, for brevity of notation, in the first component of the states we write, for instance, f_1 instead of $\mathit{\Pi}_1{\downarrow}f_1$. The strategy property we consider is $((C_{ex}, C'_{ex})) = ((\{c_2\}, \{c_1\}))$.

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 $C_i, C'_i \subseteq \mathcal{A} \cup \mathcal{K} \cup \mathcal{N}$. 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., $C_i \subseteq d(\mathcal{I})$ and $C'_i \cap d(\mathcal{I}) = \emptyset$ where $\mathcal{I}$ 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 $q \in \mathcal{G}_P$ 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 $\mathcal{T}_q$ there may exist vertices $v' \neq v$ with $\ell_V(v') = \ell_V(v)$ such that the subtrees $\mathcal{T}_q{\downarrow}v$ and $\mathcal{T}_q{\downarrow}v'$ of $\mathcal{T}_q$ 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 [11] 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 5) becomes considerably simpler.

4 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., [13, 10] for more details). In Section 5, 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 [13]. 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 $a \in \mathcal{A}_I$ 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 [7, 13, 4] 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 5 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.

We note that while standard constraint solving procedures can deal with the cryptographic primitives considered here, these procedures might need to be extended when adding further cryptographic primitives. For example, this is the case for private contract signatures, which are used in some contract signing protocols [9] and were taken

into account in [11]. However, constraint solving procedures can easily be extended to deal with these signatures. We have not considered them here for brevity of presentation and since the main focus of the present work is not to extend constraint solving procedures but to show how these procedures can be employed to deal with game-theoretic security properties.

5 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 in subsequent sections. 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 $\mathcal{T}^s_{i,q^s}$ for every (C_i, C_i') starting from this state (see Section 5.1 for more details).
2. Derive from $\mathbf{B} = \pi^s, \mathcal{T}^s_{1,q^s}, \ldots, \mathcal{T}^s_{l,q^s}$ 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 Section 5.2 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 $\mathcal{G}_P$ from the initial state of P to a state q and q-strategy trees $\mathcal{T}_{i,q}$ satisfying (C_i, C_i') for every i. If so, output yes and $\mathbf{B}$ with ν applied, and otherwise return no (see Section 5.3 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 three sections, the three steps of SolveStrategy are further explained. Our main result is the following theorem:

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

Recall that decidability of STRATEGY was already shown in [11]. The main point of Theorem 1 is that SolveStrategy uses standard constraint solving procedures as a black-box, and as such, is a good basis for extending existing practical constraint-based algorithms for reachability properties to deal with game-theoretic security properties.

The proof of Theorem 1 is quite different from the cut-and-paste argument in [11] where, similar to [14], it was shown that an attack can be turned into a "small" attack. Here we rather make use of the fact that procedures for computing sound and complete

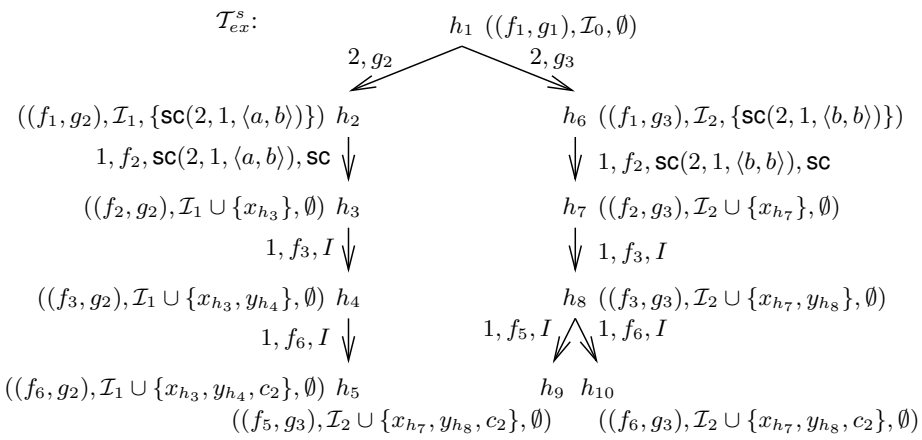

Fig. 3. Symbolic strategy tree $\mathcal{T}^s_{ex}$ for the protocol P_{ex} where $\mathcal{I}_1 = \mathcal{I}_0 \cup \{\langle a, b\rangle\}$ and $\mathcal{I}_2 = \mathcal{I}_0 \cup \{\langle b, b\rangle\}$. For brevity of notation, in the first component of the symbolic states we write, for instance, f_1 instead of $\Pi_1{\downarrow}f_1$.

solution sets exist, which makes our proof (and also our algorithm) more modular and easier to extend.

We note that if we used positional strategies as in [11], SolveStrategy would have to be extended to guess the symbolic states of symbolic branching structures that coincide after the substitution ν is applied. To avoid this, we employ the strategies with history as explained above.

5.1 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 [10] for full details). These notions will be illustrated by the example in Figure 1.

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

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

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

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 [10] for details and the example below): 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 $((\Pi_1, \ldots, \Pi_n), \mathcal{I}, \mathcal{S})$ is updated accordingly to $((\Pi'_1, \ldots, \Pi'_n), \mathcal{I}', \mathcal{S}')$ (see the example below). 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 $i, f, L', \mathtt{sc}$ 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 $\mathcal{S}'$ is obtained by removing L' from $\mathcal{S}$ 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* $\mathcal{T}^s_{q^s} = (V, E, r, \ell_V, \ell_E)$ 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.

Figure 3 depicts a symbolic q^s_0-tree $\mathcal{T}^s_{ex}$ for protocol P_{ex} (Figure 1) where $q^s_0 = (\{\Pi_1, \Pi_2\}, \mathcal{I}_0, \emptyset)$ is the symbolic initial state of P_{ex}. For brevity of notation, just as in the case of the strategy tree in Figure 1, the first component of the symbolic states in this tree does not contain the principals but only their corresponding roots. Note that the principal rules of Π_1 are applied at different places in this tree. Therefore, different copies of the variables x and y need to be introduced, which we do by indexing the variables by the name of the vertex where the rule is applied. This yields the variables $x_{h_3}, x_{h_7}, y_{h_4}, y_{h_8}$ in $\mathcal{T}^s_{ex}$.

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* $\mathcal{T}^s_{q^s} = (V, E, r, \ell_V, \ell_E)$ 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. The tree depicted in Figure 3 is a symbolic q^s_0-strategy tree.

For a protocol P and strategy property $((C_1, C'_1), \ldots, (C_l, C'_l))$, a *symbolic branching structure* is of the form $\mathbf{B}^s = \pi^s, \mathcal{T}_1{}^s, \ldots, \mathcal{T}_l{}^s$ where π^s is a symbolic path of P and the $\mathcal{T}_i{}^s$ 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.

For the strategy property $((C_{ex}, C'_{ex})) = ((\{c_2\}, \{c_1\}))$, we can consider $\mathcal{T}^s_{ex}$ in Figure 3 also as a symbolic branching structure $\mathbf{B}^s_{ex}$ of P_{ex} where the path π^s is empty and the number of symbolic strategy trees in $\mathbf{B}^s_{ex}$ is $l = 1$.

5.2 Construct and Solve the Induced Constraint System

We now show how the constraint system $\mathbf{C} = \mathbf{C_B}$ is derived from the symbolic branching structure $\mathbf{B} = \pi^s, \mathcal{T}^s_1, \ldots, \mathcal{T}^s_l$ (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. In this extended abstract, we only illustrate how $\mathbf{C}$ is derived from $\mathbf{B}$ and the strategy property by the example in Figure 1 (see [10] for full definitions).

Before turning to the example, we informally explain how to encode in a constraint system communication involving the secure channel. (Another, somewhat less interesting issue is how to deal with secure channel terms generated by the intruder. This is explained in our technical report [10].) The basic idea is that we write messages intended for the secure channel into the intruder's knowledge and let the intruder 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. Here we use the restriction on principals introduced in Section 2, namely that decryption keys can be derived by a principal. Without this condition, a principal rule of the form $\{y\}^s_x \Rightarrow x$ would be allowed even if the principal does not know (i.e., cannot derive) x. Such a rule would equip a principal with the unrealistic ability to derive any secret key from a ciphertext. Hence, the intruder, using this principal as an oracle, could achieve this as well and could potentially obtain the new keys used to encrypt messages intended for the secure channel.

We now turn to our example and explain how the (valid) constraint system, called $\mathbf{C}_{ex}$, derived from $\mathbf{B}^s_{ex}$ and $((C_{ex}, C'_{ex}))$ looks like and how it is derived from $\mathbf{B}^s_{ex}$, where $\mathbf{B}^s_{ex}$, as explained above, is simply the symbolic strategy tree $\mathcal{T}^s_{ex}$ (Figure 3): $\mathbf{C}_{ex}$ is the following sequence of constraints with an empty substitution where $k_1, k_2, k_3 \in \mathcal{A}$ are new atoms and we write $t_1, \ldots, t_n$ instead of $\{t_1, \ldots, t_n\}$.

1. $\{\langle x_{h_3}, b\rangle\}^s_{k_1} : \mathcal{I}_1, \{\langle a, b\rangle\}^s_{k_1}$
2. $\{\langle x_{h_7}, b\rangle\}^s_{k_2} : \mathcal{I}_2, \{\langle b, b\rangle\}^s_{k_2}$
3. $\{y_{h_4}\}^s_k : \mathcal{I}_1, \{\langle a, b\rangle\}^s_{k_1}, x_{h_3}$
4. $\{y_{h_8}\}^s_k : \mathcal{I}_2, \{\langle b, b\rangle\}^s_{k_2}, x_{h_7}$
5. $y_{h_4} : \mathcal{I}_1, \{\langle a, b\rangle\}^s_{k_1}, x_{h_3}, y_{h_4}$
6. $x_{h_7} : \mathcal{I}_2, \{\langle b, b\rangle\}^s_{k_2}, x_{h_7}, y_{h_8}$
7. $y_{h_8} : \mathcal{I}_2, \{\langle b, b\rangle\}^s_{k_2}, x_{h_7}, y_{h_8}$
8. $c_2 : \mathcal{I}_1, \{\langle a, b\rangle\}^s_{k_1}, x_{h_3}, y_{h_4}, c_2$
9. $c_2 : \mathcal{I}_2, \{\langle b, b\rangle\}^s_{k_2}, x_{h_7}, y_{h_8}, c_2$
10. $c_2 : \mathcal{I}_2, \{\langle b, b\rangle\}^s_{k_2}, x_{h_7}, y_{h_8}, c_2$

This constraint system is obtained from $\mathbf{B}^s_{ex}$ as follows: We traverse the vertices of $\mathbf{B}^s_{ex}$ in a top-down breadth first manner. Every edge induces a constraint except those edges which correspond to symbolic ε-transitions. This is how the constraints 1.–7. come

about where 1., 3., and 5. are derived from the left branch of $\mathbf{B}^s_{ex}$ and 2., 4., 6., and 7. from the right branch. Note that in 1. and 2. we encode the communication with the secure channel by encrypting the terms with new keys k_1 and k_2. The terms $\{\langle a, b\rangle\}^s_{k_1}$ and $\{\langle b, b\rangle\}^s_{k_2}$ are not removed anymore from the right-hand side of the constraints, i.e., from the intruder knowledge, in order for $\mathbf{C}_{ex}$ to satisfy the monotonicity property of constraint systems (recall that monotonicity is necessary for the validity of constraint systems). As explained above, since we use *new* keys and due to the restriction on principals, this does not cause problems. The constraints 8.–10. are used to ensure that c_2 can be derived at every leaf of $\mathcal{T}^s_{ex}$, a requirement that comes from our example security property $((C_{ex}, C'_{ex}))$ where $C_{ex} = \{c_2\}$. In vertex h_8 of $\mathcal{T}^s_{ex}$, two symbolic intruder transitions leave the vertex, which, as explained above, means that the associated principal rules should both be able to read the message delivered by the intruder.

Let $\mathbf{C}_1$ and $\mathbf{C}_2$ be constraint systems with empty sequences of constraints and the substitution $\nu_1 = \{x_{h_3} \mapsto a, x_{h_7} \mapsto b, y_{h_4} \mapsto a, y_{h_8} \mapsto b\}$ and $\nu_2 = \{x_{h_3} \mapsto a, x_{h_7} \mapsto b, y_{h_4} \mapsto b, y_{h_8} \mapsto b\}$, respectively. It is easy to see that $\{\mathbf{C}_1, \mathbf{C}_2\}$ is a sound and complete solution set for $\mathbf{C}_{ex}$. Since $\mathbf{C}_{ex}$ is valid, such a set can be computed by the constraint solver (Fact 1).

5.3 Check the Induced Substitutions

Let $\mathbf{B}^s = \pi^s, \mathcal{T}^s_1, \ldots, \mathcal{T}^s_l$ 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 5.2). We emphasize that for our algorithm to work, it is important that ν replaces the variables in $\mathbf{C}'$ by *new* intruder atoms from $\mathcal{A}_I$ not occurring in $\mathbf{B}^s$.

Basically, we want to check that when applying ν to $\mathbf{B}^s$, which yields $\mathbf{B}^s\nu = \pi^s\nu, \mathcal{T}^s_1\nu, \ldots, \mathcal{T}^s_l\nu$, 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 $\mathcal{G}_P$ from the initial state of $\mathcal{G}_P$ to a state $q \in \mathcal{G}_P$ and ii) $\mathcal{T}^s_i\nu$ 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 $\mathcal{G}_P$ 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 $\mathcal{T}^s_i\nu$ satisfies $(C_i, \emptyset)$. Hence, SolveStrategy only needs to make sure that 4. and 5. of Definition 1 are satisfied for every $\mathcal{T}^s_i\nu$ and that $\mathcal{T}^s_i\nu$ fulfills $(\emptyset, C'_i)$. Using that the derivation problem is decidable in polynomial time [6] (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 [10] for details).

In our example, the induced substitution for $\mathbf{C}_i$ is ν_i as $\mathbf{C}_i$ does not contain any variables. It can easily be verified that with $\mathbf{C}' = \mathbf{C}_2$ and the induced substitution ν_2, the above checks are all successful. However, they fail for $\mathbf{C}' = \mathbf{C}_1$ and ν_1 because in h_4 the rule $a \Rightarrow c_1$ could also be applied but it is not present in $\mathbf{B}^s_{ex}$. This violates Definition 1, 5. In fact, $\mathbf{B}^s_{ex}\nu_1$ would not yield a solution of the instance $(P_{ex}, ((C_{ex}, C'_{ex})))$. This example illustrates that in SolveStrategy one cannot dispense with the last step,

namely checking the substitutions, and that one has to try the different constraint systems in the sound and complete solution set for **C**.

6 Conclusion

We have shown that certain game-theoretic security properties, such as balance, of contract-signing and related protocols can be decided using standard constraint solving procedures as a black-box. This opens the way for extending existing constraint-based implementations and tools, which have successfully been employed for reachability properties, to deal with game-theoretic security properties. As future work, we plan to implement our algorithm, which will probably require some optimizations, and evaluate the algorithm on existing contract-signing and related protocols.

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 *CAV 2002*, LNCS 2404, pages 349–353. Springer, 2002.
3. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Security&Privacy 2002*, pages 86–99, 1998.
4. D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In *ESORICS 2003*, LNCS 2808, pages 253–270. Springer, 2003.
5. R. Chadha, M.I. Kanovich, and A.Scedrov. Inductive methods and contract-signing protocols. In *CCS 2001*, pages 176–185. ACM Press, 2001.
6. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *LICS 2003*, pages 261–270. IEEE, Computer Society Press, 2003.
7. Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *ASE 2001*, pages 373–376. IEEE CS Press, 2001.
8. P. H. Drielsma and S. Mödersheim. The ASW Protocol Revisited: A Unified View. In *ARSPA*, 2004.
9. J.A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO'99*, LNCS 1666, pages 449–466. Springer-Verlag, 1999.
10. D. Kähler and R. Küsters. A Constraint-Based Algorithm for Contract-Signing Protocols. Technical report, IFI 0503, CAU Kiel, Germany, 2005. Available from http://www.informatik.uni-kiel.de/reports/2005/0503.html
11. D. Kähler, R. Küsters, and Th. Wilke. Deciding Properties of Contract-Signing Protocols. In *STACS 2005*, LNCS 3404, pages 158–169. Springer, 2005.
12. S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *CSFW 2002*, pages 206–220. IEEE Computer Society, 2002.
13. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS 2001*, pages 166–175. ACM Press, 2001.
14. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theoretical Computer Science*, 299(1–3):451–475, 2003.
15. V. Shmatikov and J.C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, 2002.