ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Technical Report No. 404

An On-The-Fly Model-Checker for Security Protocol Analysis

David Basin Sebastian Mödersheim Luca Viganò

Information Security Group, ETH Zentrum, CH-8092 Zurich, Switzerland
www.infsec.ethz.ch/~{basin,moedersheim,vigano}
{basin,moedersheim,vigano}@inf.ethz.ch

April 11, 2003

Abstract

We introduce the on-the-fly model-checker OFMC, a tool that combines two ideas for analyzing security protocols based on lazy, demand-driven search. The first is the use of lazy data-types as a simple way of building efficient on-the-fly model checkers for protocols with infinite state spaces. The second is the integration of symbolic techniques and optimizations for modeling a lazy Dolev-Yao intruder, whose actions are generated in a demand-driven way. We present both techniques, along with optimizations and proofs of correctness and completeness.

Our tool is state-of-the-art both in terms of coverage and performance. For example, it finds all (but one) known attacks and discovers a new one in a test-suite of 36 protocols from the Clark/Jacob library in under one minute of CPU time for the entire suite. We also give examples demonstrating how our tool scales to, and finds errors in, large industrial-strength protocols.

1 Introduction

Model-checking, in its broadest sense, concerns finding efficient algorithms to automatically analyze properties of systems modeled as transition systems. A wide variety of model-checking approaches have recently been applied to analyzing security protocols, e.g. [1, 9, 18, 27, 28, 33]. The key challenge they face is that the general security problem is undecidable [19], and even semi-algorithms, focused on falsification, must come to terms with the enormous branching factor in the search space resulting from using the standard Dolev-Yao intruder model, where the intruder can say infinitely many different things at any point.

In this paper we show how to combine and extend different methods to build a highly effective security protocol model-checker. Our starting point is the approach of [5, 6] of using *lazy data-types* to model the infinite state-space associated with a protocol. A lazy data-type is one where data-type constructors (e.g. *cons* for building lists, or *node* for building trees) build data-types without evaluating their arguments; this allows one to represent and compute with infinite data (e.g. streams or infinite trees), generating arbitrary prefixes of the data on demand. In [5, 6], lazy data-types are used to build, and compute with, models of security protocols: a protocol and description of the powers of an intruder are formalized as an infinite tree. Lazy evaluation is used to decouple the model with search and other heuristics, building the infinite tree on-the-fly, in a demand-driven fashion.

This approach is conceptually and practically attractive as it cleanly separates model construction, search, and search reduction techniques. Unfortunately it doesn't address the problem of the prolific Dolev-Yao intruder and hence does not scale. We show how to incorporate the use of symbolic techniques to substantially reduce this problem. We formalize the *lazy intruder*, an optimization technique that significantly reduces

```
Protocol Yahalom;
Identifiers
  A,B,S: role;
  KAS,KBS,KAB: symmetric_key;
  NA,NB: nonce;
Messages
  1. A -> B: A,NA
  2. B -> S: B,{A,NA,NB}KBS
  3. S -> A: {B,KAB,NA,NB}KAS,{A,KAB}KBS
  4. A -> B: {A,KAB}KBS,{NB}KAB
Goal B authenticate S on KAB;
```

```
Protocol Yahalom;
Statistics Time: 0.02 sec
Violated_goal: B authenticate S on KAB
Attack_trace
 1. i    -> b    : i,NA
 2. b    -> i(s) : b,{i,NA,fresh(idNB,2)}kbs
 2. i(b) -> s    : b,{i,NA,fresh(idNB,2)}kbs
 3. s    -> i    : {b,kab,NA,fresh(idNB,2)}kis,
                    {i,kab}kbs
 4. i    -> b    : {i,NA,fresh(idNB,2)}kbs,
                    {fresh(idNB,2)}(NA,fresh(idNB,2))
```

Figure 1: A HLPSL-specification of the Yahalom protocol (excerpt) and the corresponding output of the OFMC tool.

the search space (i.e. the search tree) without excluding any attacks. This technique uses a symbolic representation to avoid explicitly enumerating the possible messages the Dolev-Yao intruder can generate, by representing intruder messages using terms with variables, and storing and manipulating constraints about what must be generated and from which knowledge.

Our lazy intruder is a general, technology-independent technique that can be incorporated in different approaches to protocol analysis. Here, we show how to combine it effectively with the lazy infinite-state approach to build a tool that scales well and has state-of-the-art coverage and performance. In doing so, we see our contributions as follows. First, we have extended previous approaches [22, 8, 1, 27, 21, 11, 15, 9] to the symbolic representation of the intruder so that our lazy intruder technique is applicable to a larger class of protocols and properties. Second, despite the extensions, we simplified the technique, leading to a simpler proof of its correctness and completeness. Third, the lazy intruder introduces the need for constraint reduction and this introduces its own search space. We have formalized the integration of the technique into the search procedure induced by the rewriting approach of our underlying protocol model, which provides an infinite-state transition system. On the practical side, we have also investigated the question of an efficient implementation of the lazy intruder, i.e. how to organize state exploration and constraint reduction.

The result is the on-the-fly model-checker for security protocol analysis OFMC. We have carried out a large number of experiments to validate our approach. For example, the OFMC tool finds all (but one) known attacks and discovers a new one (on the Yahalom protocol) in a test-suite of 36 protocols from the Clark/Jacob library [13] in under one minute of CPU time for the entire suite. Moreover, we have been applying OFMC to large-scale protocols including IKE, SET, and various other industrial protocols being currently standardized by the Internet Engineering Task Force IETF. As an example of industrial-scale problem, we describe in §5.2 our analysis of the H.530 protocol [23], a protocol invented by Siemens and proposed as an Internet standard for multimedia communications. We have modeled the protocol in its full complexity and have detected a replay attack in 1.6 seconds. The weakness is serious enough that Siemens has changed the protocol.

The remainder of this paper is organized as follows. In §2 we give the formal model that we use for protocol analysis. In §3 we briefly review the lazy protocol analysis approach. In §4 we formalize the lazy intruder and how constraints are reduced. We present experimental results in §5, discuss related work in §6, and draw conclusions and discuss future work in §7.

Due to lack of space, examples and proofs have been shortened or moved to the appendix.

2 Protocol Specification Languages and Model

The formal model we use for the protocol analysis with our tool OFMC is based on two specification languages: a high-level language (HLPSL) and a low-level one (IF). As most of the ideas behind the HLPSL are standard, e.g. [16, 24], we explain its main features on an example. Figure 1 shows the core of an HLPSL-specification of the Yahalom protocol and the trace of a new attack that we have found, as output by OFMC, which we discuss in more detail in §A.

The Yahalom protocol aims at distributing a session key K_{AB} to two agents playing in the roles A and B with the help of a trusted server playing in the role S. We specify the list of messages exchanged between the agents acting in the protocol roles, which in this case are A, B and S, and one security property. We also specify information that is often left implicit (or that is explained informally) in protocol declarations. For instance, we specify the types of the identifiers used, which determines their properties. In the example we declare symmetric keys and nonces that are generated during protocol execution. HLPSL also supports the specification of the initial knowledge of the agents and of possible intruder behavior (e.g. variants of the Dolev-Yao model), as well as asymmetric encryption, cryptographic hash-functions, non-atomic keys, key-tables, and Diffie-Hellman key-agreement.

A translator called HLPSL2IF automatically translates a high-level specification into a low-level *intermediate format IF* based on first-order (multi)set rewriting. The IF unambiguously specifies an *infinite state transition system*, and thus provides a general formalism for modeling security protocols.[1] Our tool OFMC works at this level, which means that the approach and the methods we present are independent from the question of how one encodes a protocol as transition rules.

2.1 The Syntax of the IF

Definition 1. *Let $\mathcal{C}$ and $\mathcal{V}$ be disjoint countable sets of* constants *(denoted by lower-case letters) and* variables *(denoted by upper-case letters). The* syntax *of the IF is defined by the following context-free grammar:*

$$
\begin{array}{rcl}
\mathit{ProtocolDescr} & ::= & (\mathit{State}, \mathit{Rule}^*, \mathit{State}^*) \\
\mathit{Rule} & ::= & \mathit{State}\ \mathit{NegFact} \Rightarrow \mathit{State} \\
\mathit{State} & ::= & \mathit{PosFact}\ (\ .\ \mathit{PosFact})^* \\
\mathit{NegFact} & ::= & (\ .\ \mathsf{not}(\mathit{PosFact})\ )^* \\
\mathit{PosFact} & ::= & \mathsf{state}(\mathit{Msg}) \mid \mathsf{message}(\mathit{Msg}) \mid \mathsf{i_knows}(\mathit{Msg}) \mid \mathsf{secret}(\mathit{Msg}, \mathit{Msg}) \\
\mathit{Msg} & ::= & \mathit{AtomicMsg} \mid \mathit{ComposedMsg} \\
\mathit{ComposedMsg} & ::= & \langle \mathit{Msg}, \mathit{Msg} \rangle \mid \{\mathit{Msg}\}_{\mathit{Msg}} \mid \{|\mathit{Msg}|\}_{\mathit{Msg}} \mid \mathit{Msg}^{-1} \\
\mathit{AtomicMsg} & ::= & \mathcal{C} \mid \mathcal{V} \mid \mathbb{N} \mid \mathsf{fresh}(\mathcal{C}, \mathbb{N})
\end{array}
$$

We write $\mathcal{L}(N)$ for the context-free language associated to the non-terminal N. We write $\mathrm{vars}(t)$ *to denote the set of variables occurring in a (message, fact, or state)* term t*, and say that t is* ground *when* $\mathrm{vars}(t) = \emptyset$.

An *atomic message* is a constant, a variable, a natural number, or a *fresh constant*. The fresh constants are used to model the creation of random data, e.g. nonces, during a protocol session. We model each fresh data item by a unique term $\mathsf{fresh}(C,N)$, where C is an identifier in the HLPSL specification and the number N denotes the particular protocol session C is intended for.

Messages in the IF are either atomic messages or are composed using *pairing* $\langle M_1, M_2 \rangle$, or the *cryptographic operators* $\{M_1\}_{M_2}$ and $\{|M_1|\}_{M_2}$ (for *asymmetric* and *symmetric encryption* of M_1 with M_2), or M^{-1} (the *asymmetric inverse* of M).[2] Note that we follow the standard *perfect cryptography assumption*, i.e. the only way to decrypt an encrypted message is to have the appropriate key. Moreover, like most other approaches, we employ the *free algebra assumption* and assume that syntactically different terms represent different messages, facts, or states. In other words, we do not assume that algebraic equations hold on terms, e.g. that pairing is associative.[3] Note too that unlike other models, e.g. [17, 27], we are not bound to a fixed public-key infrastructure where every agent initially has a key-pair and knows the public key of every other agent. Rather, we can also consider protocols where keys are generated, distributed, and revoked.

[1] By default the IF is untyped, but it can also be generated in a typed variant, which leads to smaller search spaces at the cost of abstracting away the type-flaw attacks (if any) on the protocol.

[2] Some approaches, e.g. [29] extend the function $\cdot^{-1}$ to symmetric keys (in this case, the inverse key is identical). We cannot do this since in our model messages are untyped, so the inverse key cannot be determined from the type of key. In our model, every message has an asymmetric inverse. As we will define, cf. Def. 3, the intruder (as well as the honest agents) can compose a message from its submessages but cannot generate M^{-1} from M. The only ways to obtain the inverse of a message is to know it initially, to receive it in a message, or it is the private key of a self-generated asymmetric key-pair.

[3] In our model, $(M^{-1})^{-1} = M$ is respected while the free algebra assumption is preserved: as no agent, not even the intruder, can generate M^{-1} from M, we ensure that $(M^{-1})^{-1}$ is never produced by having two rules for the analysis of asymmetric encryptions, one for public keys and one for private ones.

The IF contains both positive and negative *facts*. A *positive fact* represents either the local state of an honest agent, a message on the network (i.e. one sent but not yet received), that a message is known by the intruder, or that a message is a secret, where $\mathsf{secret}(\mathsf{a},\mathsf{m})$ means that m is a secret that may be known by agent a. This allows us, for instance, to specify as a goal that m is known only by the specified agents, and that it is not the case that the intruder knows m (i.e. $\mathsf{i_knows}(\mathsf{m})$) unless $\mathsf{secret}(\mathsf{i},\mathsf{m})$ is explicitly specified. *Negative facts* allow us to formalize this, as they allow us to express goals that explicitly require negation, e.g. to state that the intruder does not find out some secret. Moreover, negative facts allow for the modeling of a wider range of protocols than with a language with standard rewrite rules (i.e. rules that contain only positive facts). For example, we can model a protocol where an agent stores nonces for replay detection and accepts messages only if they contain nonces that he has not previously generated or received.

A *state* is a set of positive facts, which we denote as a sequence of positive facts separated by dots. Note that in our approach we actually employ set rewriting instead of multiset rewriting, as the HLPSL2IF translator ensures that in no reachable state the same positive fact can appear more than once, so we do not need to distinguish between multisets and sets. Note also that the sets of positive facts and composed messages (i.e. the context-free languages $\mathcal{L}(\mathit{Fact})$ and $\mathcal{L}(\mathit{ComposedMsg})$) can be easily extended, e.g. with cryptographic primitives for hashes and key-tables, without affecting the theoretical results we present below. Here we focus on this smaller language for brevity.

A *protocol description ProtocolDescr* is a triple (I, R, G), consisting of an *initial state* I, a *set of rules* R, and a *set of goal states* G. A protocol description constitutes a *protocol* when two restrictions are met: (i) the initial state is ground, and (ii) $\mathrm{vars}(l_1) \supseteq \mathrm{vars}(l_2) \cup \mathrm{vars}(r)$ for every rule $l_1.l_2 \Rightarrow r$ in R where l_1 contains only positive facts and l_2 contains only negative facts.

Before we consider rules in detail, let us extend some standard concepts [4].

Definition 2. *A* substitution σ *is a mapping from* $\mathcal{V}$ *to* $\mathcal{L}(\mathit{Msg})$. *The* domain *of* σ*, denoted by* $\mathrm{dom}(\sigma)$*, is the set of variables* $V \subseteq \mathcal{V}$ *such that* $\sigma(v) \neq v$ *iff* $v \in V$. *In this paper, we only consider substitutions with finite domains and represent a substitution* σ *with* $\mathrm{dom}(\sigma) = \{v_1, \ldots, v_n\}$ *by* $[v_1 \mapsto \sigma(v_1), \ldots, v_n \mapsto \sigma(v_n)]$. *The* identity substitution id *is the substitution with* $\mathrm{dom}(\mathrm{id}) = \emptyset$. *We say that a substitution* σ *is* ground*, and write* $\mathrm{ground}(\sigma)$*, if* $\sigma(v)$ *is a ground term for all* $v \in \mathrm{dom}(\sigma)$. *We extend* σ *to a homomorphism on message terms, facts, and states in the standard way, and also write* $t\sigma$ *for* $\sigma(t)$. *We say two substitutions* σ_1 *and* σ_2 *are* compatible*, written* $\sigma_1 \approx \sigma_2$*, if* $v\sigma_1 = v\sigma_2$ *for every* $v \in \mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2)$. *The* composition *of* σ_1 *and* σ_2 *is denoted as* $\sigma_1\sigma_2$. *Note that* $\sigma_1\sigma_2 = \sigma_2\sigma_1$ *for compatible ground substitutions. For two sets of ground substitutions* Σ_1 *and* Σ_2*, we define their* intersection modulo the different domains *as*

$$\Sigma_1 \sqcap \Sigma_2 = \{\sigma_1\sigma_2 \mid \sigma_1 \in \Sigma_1 \ \wedge \ \sigma_2 \in \Sigma_2 \ \wedge \ \sigma_1 \approx \sigma_2\}.$$

Two terms unify *when there exists a substitution, called their* unifier*, under which they are equal.* Matching *is the special case where one of the terms is ground. Since we are working under the free algebra assumption, two unifiable terms always have a* most general unifier *(mgu).*

Rules describe state transitions. Intuitively, the application of the rule $l_1.l_2 \Rightarrow r$ means that if there is a substitution σ such that no positive fact f such that $\mathsf{not}(f) \in l_2$ can be matched under σ with the current state, and all positive facts in l_1 can be matched under σ with the current state, then $l_1\sigma$ is replaced by $r\sigma$ in the current state. Otherwise, the rule is not applicable.

In this paper, we consider only IF rules of the form (which allows us to analyze a large number of protocols, including those discussed in §5)

$$\mathsf{message}(m_1).\mathsf{state}(m_2).P_1.N_1 \Rightarrow \mathsf{state}(m_3).\mathsf{message}(m_4).P_2\,, \tag{1}$$

where N is a set of negative facts, and P_1 and P_2 are sets of positive facts that do not contain state or $\mathsf{message}$ facts. Moreover, if $\mathsf{i_knows}(m) \in P_1$ then $\mathsf{i_knows}(m) \in P_2$, which ensures that the intruder knowledge is *monotonic*, i.e. that the intruder never forgets messages during a transition.

More specifically, every rule describes a transition of an honest agent, since a state fact appears on both the left-hand side (LHS) and the right-hand side (RHS). Also, on both sides we have a $\mathsf{message}$ fact representing an incoming message that the agent expects to receive in order to make the transition (LHS) and an answer message from the agent (RHS). The rule corresponding to the initial (respectively, final)

protocol step contains no incoming (respectively, outgoing) message, but the rule form (1) is in this case not a restriction, as one may always insert a dummy message that can be generated by the intruder. An example of an actual IF rule corresponding to one of the messages of the Yahalom protocol is given in §A.2.

2.2 The Dolev-Yao Intruder

We follow Dolev and Yao [17] and consider the standard, protocol-independent, asynchronous model in which the intruder controls the network but cannot break cryptography. In particular, the intruder can intercept messages and analyze them if he possesses the corresponding keys for decryption, and he can generate messages from his knowledge and send them under any agent name.

Definition 3. *For a set M of messages, let $\mathcal{DY}(M)$ (for Dolev-Yao) be the smallest set closed under the following* generation (G) *and* analysis (A) rules*:*

$$\frac{m \in M}{m \in \mathcal{DY}(M)}\ G_{\text{axiom}}\,,\quad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\langle m_1,m_2\rangle \in \mathcal{DY}(M)}\ G_{\text{pair}}\,,\quad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{m_2\}_{m_1} \in \mathcal{DY}(M)}\ G_{\text{crypt}}\,,$$

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{\!|m_2|\!\}_{m_1} \in \mathcal{DY}(M)}\ G_{\text{scrypt}}\,,\quad \frac{\langle m_1,m_2\rangle \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)}\ A_{\text{pair}_i}\,,\quad \frac{\{\!|m|\!\}_k \in \mathcal{DY}(M) \quad k \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\ A_{\text{scrypt}}\,,$$

$$\frac{\{m\}_k \in \mathcal{DY}(M) \quad k^{-1} \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\ A_{\text{crypt}}\,,\quad \frac{\{m\}_{k^{-1}} \in \mathcal{DY}(M) \quad k \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\ A_{\text{crypt}^{-1}}\,.$$

The generation rules express that the intruder can compose messages from known messages using pairing and asymmetric and symmetric encryption. The analysis rules describe how the intruder can decompose messages. Note that this formalization correctly handles non-atomic keys, e.g. $m \in \mathcal{DY}(\{\{\!|m|\!\}_{(\langle k_1,k_2\rangle)}, k_1, k_2\})$, as opposed to other models such as [29] that only handle atomic keys.

2.3 The Semantics of the IF

Using $\mathcal{DY}$, we can now define the protocol model provided by the IF in terms of an infinite-state transition system, where the IF rules define a state transition function. In this definition, we incorporate an optimization that we call *step-compression*, which is based on the idea [1, 8, 12, 15, 27] that we can identify the intruder and the network: every message sent by an honest agent is received by the intruder and every message received by an honest agent comes from the intruder. Formally, we compose (or "compress") several steps: when the intruder sends a message, an agent reacts to it according to his rules, and the intruder diverts the agent's answer. A bisimulation proof shows that the model with such composed actions (which we present here) is "attack-equivalent" to the model with single (uncompressed) transitions (i.e. we end up in an attack state using composed transitions iff that was the case using uncomposed transitions).

Definition 4. *The* successor function *maps a set of rules R and a state S to a set of states:*

$$\begin{aligned}
&succ_R(S) = \bigcup_{r \in R} step_r(S)\\
&step_r(S) = \{S' \mid \exists\ \sigma.\\
&\qquad \text{ground}(\sigma)\ \wedge\ \text{dom}(\sigma) = \text{vars}(m_1) \cup \text{vars}(m_2) \cup \text{vars}(P_1)\ \wedge && (2)\\
&\qquad m_1\sigma \in \mathcal{DY}(\{\ i \mid \mathsf{i_knows}(i) \in S\})\ \wedge && (3)\\
&\qquad \mathsf{state}(m_2\sigma) \in S\ \wedge\ P_1\sigma \subseteq S\ \wedge && (4)\\
&\qquad \forall f.\ \mathsf{not}(f) \in N_1\ \Longrightarrow\ f\sigma \notin S\ \wedge && (5)\\
&\qquad S' = (S \setminus (\mathsf{state}(m_2\sigma) \cup P_1\sigma)) \cup \mathsf{state}(m_3\sigma) \cup \mathsf{i_knows}(m_4\sigma) \cup P_2\sigma\} && (6)\\
&\quad \text{for a rule } r \text{ of the form } \mathsf{message}(m_1).\mathsf{state}(m_2).P_1.N_1 \Rightarrow \mathsf{state}(m_3).\mathsf{message}(m_4).P_2
\end{aligned}$$

Here and elsewhere, we simplify notation for singletons, writing, e.g., $\mathsf{state}(m_2\sigma) \cup P_1\sigma$ *for* $\{\mathsf{state}(m_2\sigma)\} \cup P_1\sigma$*.*

The step function implements the step-compression technique described above in that it combines three actions, based on a rule r of the form (1). The three actions are: the intruder sends a message that is expected by an honest agent, the honest agent receives the message and sends a reply, and the intruder diverts this reply and adds it to his knowledge. More in detail, the condition (3) ensures that the message $m_1\sigma$ (that is expected by the honest agent) can be generated from the intruder knowledge under σ, where according to (2) σ is a ground substitution for the variables in the positive facts of the LHS of the rule r. The conjuncts (4) ensure that the other positive facts of the rule appear in the current state under σ, and (5) ensures that none of the negated facts is contained in the current state under σ. Finally, (6) defines the successor state S' that results by removing from S the positive facts of the LHS of r and replacing them with the RHS of r (all under σ).

Definition 5. *We define the* set of reachable states *associated to a protocol* (I, R, G) *as* $reach(I, R) = \bigcup_{n\in\mathbb{N}} succ_R^n(I)$.

The set of reachable states is ground as no state reachable from the initial state I may contain variables (by the conditions (i) and (ii) in the definition of protocol). As the properties we are interested in are reachability properties, we will sometimes abstract away the details of the transition system and refer to this set as the *ground model* of the protocol.

Definition 6. *We say that a protocol is* secure *iff* $goalcheck_g(S) = \emptyset$ *for all* $S \in reach(I, R)$ *and all goals* $g \in G$, *where we define* $goalcheck_g(S) = \{\sigma \mid g\sigma \subseteq S\}$.

3 The Lazy Infinite-State Approach

The transition system defines a (computation) tree in the standard way, where the root is the initial system state, and children represent the ways that a state can evolve in one transition. The tree has infinitely many states. By the definition of $\mathcal{DY}$, every node has infinitely many children. It is also of infinite depth, provided we do not bound (and we cannot recursively bound) the number of interleaved protocol sessions. The lazy intruder technique presented in the next section provides a way to tackle the infinite branching using a symbolic representation, while the *lazy infinite-state approach* [5, 6] allows us to handle the infinitely long branches. As we have integrated the lazy intruder with our previous work, we now briefly summarize the main ideas of [5, 6].

The key idea behind the lazy infinite-state approach is to explicitly formalize an infinite tree as an element of a data-type in a lazy programming language. This yields a finite, computable representation of the model that can be used to generate arbitrary prefixes of the tree on-the-fly, i.e. in a demand-driven way. One can search for an attack by searching the infinite tree for a goal state. Our on-the-fly model-checker OFMC uses iterative deepening to search this infinite tree for an attack state. When an attack is found, OFMC reports the attack trace, i.e. the sequence of exchanged messages along the path from the initial state to the attack state (cf. Fig. 1 and §A.1). This yields a semi-decision procedure for protocol insecurity: our procedure always terminates (at least in principle) when an attack exists. Moreover, our search procedure terminates for finitely many sessions (formally: if there are finitely many agents and none of them can perform an unbounded number of transitions), and this even when we do not employ the lazy intruder to restrict the infinite set of messages the intruder can generate.

The lazy approach has several strengths. In particular, it separates (both conceptually and structurally) the semantics of protocols from heuristics and other search reduction procedures, and from search itself. The semantics is given by a program generating an infinite tree, and heuristics can be seen as tree transducers that take an infinite tree and return one that is, in some way, smaller or more restricted. The resulting tree can then be searched. Although semantics, heuristics, and search can all be formulated independently, lazy evaluation serves to co-routine them together in an efficient, demand-driven fashion. Moreover there are efficient compilers for lazy functional programming languages like Haskell, the language we used.

4 The Lazy Intruder

The *lazy intruder* is an optimization technique that significantly reduces the search tree without excluding any attacks. This technique uses a symbolic representation to avoid explicitly enumerating the possible messages that the Dolev-Yao intruder can generate, by storing and manipulating constraints about what must be generated (cf. §4.1 and §4.2). The representation is evaluated in a demand-driven way, hence the intruder is called *lazy*.

The idea behind the lazy intruder was, to our knowledge, first proposed by [22] and subsequently developed by [22, 8, 1, 27, 21, 11, 15, 9] Our contributions to the symbolic intruder technique are:

- We have simplified the technique, leading to a simpler proof of its correctness and completeness (cf. §4.2–4.4, e.g. Lemma 3, and §B).
- We have formalized its integration into the search procedure induced by the rewriting approach of the IF (cf. §4.4). On the practical side, we have investigated the question of efficient implementation, i.e. how to organize state exploration and constraint reduction (cf. §4.5).
- We have extended the technique so that it is applicable to a larger class of protocols and properties: as we show in §4.4, the introduction of negative facts alongside standard positive facts in the IF rewrite rules, providing us with a more expressive specification language, leads to inequality constraints in the lazy intruder.

4.1 Constraints

The Dolev-Yao intruder leads to an enormous branching of the search tree when one naïvely enumerates all (meaningful) messages that the intruder can send. The lazy intruder technique exploits the fact that the actual value of certain parts of a message is often irrelevant for the receiver. So, whenever the receiver will not further analyze the value of a particular message part, we can postpone during the search the decision about which value the intruder actually chooses for this part by replacing it with a variable and recording a *constraint* on which knowledge the intruder uses to generate the message. We express this information using constraints of the form $\mathit{from}(T, IK)$, meaning that T is a set of terms generated by the intruder from his set of known messages IK (for "intruder knowledge").

As an example, consider again the Yahalom protocol and suppose that the intruder i impersonates a server s and sends a forged instance of the third message to an agent a playing in the role A. Then i can send

$$3.\ \mathsf{i}(\mathsf{s}) \to \mathsf{a}:\ \{|\mathbf{b}, m_1, \mathbf{n_a}, m_2|\}_{\mathbf{k_{as}}}, m_3$$

with the constraint $\mathit{from}((\{|\mathbf{b}, m_1, \mathbf{n_a}, m_2|\}_{\mathbf{k_{as}}}, m_3), IK)$ containing the variable message parts m_1, m_2, and m_3. a will accept this message since the only parts for which a expects a particular value (displayed using bold-face) are $\mathbf{b}$ and $\mathbf{k_{as}}$, as well as $\mathbf{n_a}$, which a generated himself when he sent the first message of the protocol. This constraint is stored and further instantiated during the search.

Definition 7. *The* semantics of a constraint $\mathit{from}(T, IK)$ *is the set of satisfying ground substitutions σ for the variables in the constraint, i.e.*

$$[\![\mathit{from}(T, IK)]\!] = \{\sigma \mid \mathrm{ground}(\sigma) \wedge \mathrm{ground}(T\sigma \cup IK\sigma) \wedge (T\sigma \subseteq \mathcal{DY}(IK\sigma))\}\,.$$

A constraint set *is a finite set of constraints, and its semantics is the intersection of the semantics of its elements, i.e., overloading the $[\![\ldots]\!]$ notation, $[\![\{c_1, \ldots, c_n\}]\!] = \sqcap_{i=1}^{n} [\![c_i]\!]$. A constraint set C is* satisfiable *if $[\![C]\!] \neq \emptyset$. A constraint $\mathit{from}(T, IK)$ is* simple *if $T \subseteq \mathcal{V}$, and we then write* $\mathrm{simple}(\mathit{from}(T, IK))$. *A constraint set is simple if all its constraints are simple. We overload* simple *and apply it to constraint sets as expected.*

4.2 Constraint Reduction

The core of the lazy intruder technique is to reduce a given constraint set into an equivalent one that is either unsatisfiable or simple. (As we show in Lemma 3, every simple constraint set is satisfiable.) This reduction

$$\frac{\mathit{from}(m_1 \cup m_2 \cup T, IK) \cup C, \sigma}{\mathit{from}(\langle m_1,m_2\rangle \cup T, IK) \cup C, \sigma}\ G^l_{\text{pair}}\,, \quad \frac{\mathit{from}(m_1 \cup m_2 \cup T, IK) \cup C, \sigma}{\mathit{from}(\{\!|m_2|\!\}_{m_1} \cup T, IK) \cup C, \sigma}\ G^l_{\text{scrypt}}\,,$$

$$\frac{(\mathit{from}(T, m_2 \cup IK) \cup C)\tau, \sigma\tau}{\mathit{from}(m_1 \cup T, m_2 \cup IK) \cup C, \sigma}\ G^l_{\text{unif}}\ (\tau = mgu(m_1, m_2),\ m_1 \notin \mathcal{V})\,,$$

$$\frac{\mathit{from}(T, m_1 \cup m_2 \cup \langle m_1,m_2\rangle \cup IK) \cup C, \sigma}{\mathit{from}(T, \langle m_1,m_2\rangle \cup IK) \cup C, \sigma}\ A^l_{\text{pair}}\,, \quad \frac{\mathit{from}(k, IK) \cup \mathit{from}(T, m \cup \{\!|m|\!\}_k \cup IK) \cup C, \sigma}{\mathit{from}(T, \{\!|m|\!\}_k \cup IK) \cup C, \sigma}\ A^l_{\text{scrypt}}\,.$$

Figure 2: Lazy intruder: constraint reduction rules

is performed using the generation and analysis rules of Fig. 2, which describe possible transformations of the constraint set (for brevity, we have omitted the rules for asymmetric encryption and decryption, which are straightforward). Afterwards, we show that this reduction does not change the set of solutions, roughly speaking $[\![C]\!] = [\![\mathit{Red}(C)]\!]$, for a relevant class of constraints C.

The rules are of the form

$$\frac{C', \sigma'}{C, \sigma}\ ,$$

with C and C' constraint sets and σ and σ' substitutions, and they express that (C', σ') can be *derived* from (C, σ), which we denote by $(C, \sigma) \vdash (C', \sigma')$. Note that in all rules σ' extends σ. As a result, we can apply the substitutions generated during the reduction of C also to the facts of the lazy state.

The generation rules G^l_{pair} and G^l_{scrypt} express that the constraint stating that the intruder can generate a message composed from submessages m_1 and m_2 using pairing or symmetric encryption can be replaced by the constraint stating that he can generate both m_1 and m_2. The rule G^l_{unif} expresses that the intruder can use a message m_2 from his knowledge if this message can be unified with the message m_1 he has to generate (note that both the terms to be generated and the terms in the intruder knowledge may contain variables). The reason that the intruder is "lazy" stems from the restriction that the G^l_{unif} rule cannot be applied when the term to be generated is a variable: the intruder's choice for this variable does not matter at this stage of the search and hence we postpone this decision.

The analysis of the intruder knowledge is more complex for the lazy intruder than in the ground model, as messages may now contain variables. In particular, if the key of an encrypted message is a variable, then whether or not the intruder can decrypt this message is determined by the substitution we (later) choose for this variable. We solve this problem by using the rule A^l_{scrypt}, where the variable key can be instantiated during further constraint reduction.[4] More specifically, for a message $\{\!|m|\!\}_k$ that the intruder attempts to decrypt, we add the content m to the intruder knowledge of the respective constraint (as if the check was already successfull) and add a new constraint expressing that the symmetric key k necessary for decryption must be generated from the same knowledge. Hence, if we attempt to decrypt a message that cannot be decrypted using the corresponding intruder knowledge, we obtain an unsatisfiable constraint set.

Note that we also make the restriction that the message to be analyzed may not be used in the generation of the key, in contrast to similar approaches that can handle non-atomic symmetric keys, e.g. [27, 12]. In our notation, their decryption rule is:

$$\frac{\mathit{from}(k, \{\!|m|\!\}_k{}^* \cup IK) \cup \mathit{from}(T, m \cup \{\!|m|\!\}_k \cup IK) \cup C, \sigma}{\mathit{from}(T, \{\!|m|\!\}_k \cup IK) \cup C, \sigma}\ A^l_{\text{scrypt}}{}^*\,.$$

The rule is the same as ours, except that the constraint of the derivation of the key k additionally contains the message $\{\!|m|\!\}_k$ marked with a $*$. The message is marked to denote that it may not be further analyzed (as there is already an analysis of this term in progress). Otherwise, i.e. without the mark in the constraint, the approaches of [27, 12] would not terminate. (This is because they use sequences of constraints instead of sets like in our approach, so that in the derivation of k one would try infinitely often to first decrypt $\{\!|m|\!\}_k$ repeatedly producing the same constraint.) However, the addition of the mark gives the rule a procedural

[4] This solution also takes care of non-atomic keys since we do not require that the key is contained in the intruder knowledge but only that it can be generated from the intruder knowledge (e.g. by composing known messages).

aspect and makes it less declarative. As formally justified in the proof of the completeness theorem (the proof of Theorem 1 in §B), our rule A^l_{scrypt}, which omits the marked term entirely, does not exclude any solution. The intuition behind this is as follows: the only case in which the marked term $\{|m|\}_k$ is actually used to derive k is when there is a further term $t \in IK$ that is encrypted with the term $\{|m|\}_k$ as a key. In this case, however, we could have first performed the analysis of t so that we do not need to perform it during the derivation of k. In general, if one performs the analysis steps in the sequence as they depend on each other, no analysis is needed in the constraints that are introduced by the analysis rules (in this case the $\mathit{from}(k, \cdot)$ constraint).

Note that our rule is not only simpler and more declarative, but it also considerably simplifies the completeness proof. (For example, the respective completeness proof in [27] is split into a part with *encryption hiding* (as they call the marked terms) and one without.)

Definition 8. *Let $\vdash$ denote the derivation relation described by the rules in Fig. 2. The set of pairs of simple constraint sets and substitutions that can be derived from (C, id) is $\mathit{Red}(C) = \{(C', \sigma) \mid ((C, \mathrm{id}) \vdash (C', \sigma)) \wedge \mathrm{simple}(C')\}$.*

4.3 Properties of *Red*

We show below that the *Red* function is correct and complete. Moreover we show that the set of constraint sets $\mathit{Red}(C)$ is finite for every C and is recursively computable. To show completeness, we restrict our attention to a special form of constraint sets, called *well-formed constraint sets*. This is without loss of generality, as all states reachable in the lazy intruder setting obey this restriction (cf. Def. 9 and Lemma 4).

Definition 9. *A constraint set C is* well-formed *if one can index the constraints, $C = \{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_n, IK_n\})$, so that the following conditions hold:*

$$IK_i \subseteq IK_j \text{ for } i \leq j\,, \tag{7}$$

$$\mathrm{vars}(IK_i) \subseteq \cup_{j=1}^{i-1} \mathrm{vars}(T_j)\,. \tag{8}$$

Intuitively, (7) requires that the intruder knowledge increases monotonically, and (8) requires that every variable that appears in intruder-known terms is part of a message that the intruder created earlier, i.e. variables only "originate" from the intruder.

Note that the analysis rules of the lazy intruder can destroy property (7), as the message obtained by an analysis rule is not necessarily contained in the subsequent (i.e. of higher index) intruder knowledge sets. However, as we show formally in the proof of Theorem 1, there is a straightforward procedure that transforms every simple constraint set obtained by *Red* into an equivalent, well-formed, simple one.

The main results about *Red* states that the reduction of a finite constraint set C always terminates and results in a finite set of simple constraint sets that together have the same set of solutions as C; the proofs can be found in §B.

Lemma 1 (Termination of *Red*). *$\mathit{Red}(C)$ is finite and recursively computable for every constraint set C.*

Theorem 1 (Correctness and completeness of *Red*). *$[\![C]\!] = \{\sigma\sigma' \mid (C', \sigma) \in \mathit{Red}(C) \wedge \sigma' \in [\![C']\!]\}$ for a well-formed constraint-set C.*

4.4 The Lazy Intruder Reachability

We describe now the integration of constraint reduction into the search procedure for reachable states. The space of *lazy states* consists of states that may contain variable symbols (as opposed to the ground model where all reachable states are ground) and that are associated to a set of *from* constraints as well as a collection of inequalities. The inequalities will be used to handle negative facts in the context of the lazy intruder. We assume that the inequalities are given as a conjunction of disjunctions of inequalities between terms. We will use the inequations to rule out certain unifications, e.g. to express that both the substitutions $\sigma = [v_1 \mapsto t_1, v_2 \mapsto t_2]$ and $\tau = [v_1 \mapsto t_3]$ are excluded in a certain state, we use the inequality constraint $(v_1 \neq t_1 \mathbin{\mathsf{V}} v_2 \neq t_2) \mathbin{\mathsf{\Lambda}} (v_1 \neq t_3)$, where we use V and $\mathsf{\Lambda}$ to avoid confusion with the respective meta-connectives $\vee$ and $\wedge$.

A lazy state represents the set of ground states that can be obtained by instantiating the variables with ground messages so that all associated constraints are satisfied:

Definition 10. *A* lazy state *is a triple (P, C, N), where P is a sequence of (not necessarily ground) positive facts, C is a constraint set, and N is a conjunction of disjunctions of inequalities between terms. The semantics of a lazy state is*

$$[\![(P, C, N)]\!] = \{P\sigma \mid \sigma \in [\![C]\!] \wedge \sigma \models N\},$$

where we define $\sigma \models N$ for a substitution σ as expected.

Let $\mathrm{freshvars}_r(S)$ *be a renaming of the variables in a lazy state $S = (P, C, N)$ with respect to a rule r such that* $\mathrm{vars}(\mathrm{freshvars}_r(S))$ *and* $\mathrm{vars}(r)$ *are disjoint. (This can be achieved by defining a total order on the set of variable symbols $\mathcal{V}$ and whenever a new variable is needed, the smallest symbol not used in r under this order is chosen.) The* lazy successor function *maps a set of rules R and a lazy state $S = (P, C, N)$ to a set of lazy states:*

$$\begin{aligned}
&\mathit{lsucc}_R(S) = \cup_{r \in R}\ \mathit{lstep}_r(\mathrm{freshvars}_r(S))\\
&\mathit{lstep}_r(P, C, N) = \{(P', C', N') \mid \exists\ \sigma.\\
&\qquad \mathrm{dom}(\sigma) \subseteq \mathrm{vars}(m_1) \cup \mathrm{vars}(m_2) \cup \mathrm{vars}(P_1) \cup \mathrm{vars}(P1) \cup \mathrm{vars}(C1) \cup \mathrm{vars}(N)\ \wedge\\
&\qquad C' = (C \cup \mathit{from}(m_1, \{i \mid \mathsf{i_knows}(i) \in P\}))\sigma\ \wedge \qquad (9)\\
&\qquad \mathsf{state}(m_2\sigma) \in P\sigma\ \wedge P_1\sigma \in P\sigma\ \wedge \qquad (10)\\
&\qquad N' = N \wedge \textstyle\bigwedge_{\phi \in \mathit{subCont}(N_1\sigma, P\sigma)}\ \phi\ \wedge \qquad (11)\\
&\qquad P' = (P\sigma \setminus (\mathsf{state}(m_2\sigma) \cup P_1\sigma))\ \cup\ \mathsf{state}(m_3\sigma) \cup \mathsf{i_knows}(m_4\sigma) \cup P_2\sigma\} \qquad (12)
\end{aligned}$$

for a rule r of the form $\mathsf{message}(m_1).\mathsf{state}(m_2).P_1.N_1 \Rightarrow \mathsf{state}(m_3).\mathsf{message}(m_4).P_2$

$$\begin{aligned}
&\mathit{subCont}(N, P) = \{\phi \mid \exists\ t, t', \sigma.\ \mathsf{not}(t) \in N\ \wedge\ t' \in P\ \wedge\ t\sigma = t'\sigma\ \wedge\\
&\qquad \exists\ v_1, \ldots, v_n, t_1, \ldots, t_n.\ \sigma = [v_1 \mapsto t_1, \ldots, v_n \mapsto t_n]\ \wedge\ \phi = \textstyle\bigvee_{i=1}^{n}\ v_i \neq t_i\}
\end{aligned}$$

Similar to the successor function of the ground model (Def. 5), the lazy successor function also performs three operations in one transition: the intruder sends a message, an honest agent reacts to it, and the intruder adds the answer to his knowledge. The most notable change is the renaming of the variables of the rules to avoid clashes with the variables that may appear in the lazy states. More in detail, the constraint in condition (9) expresses that the message m_1 that occurs on the LHS of the rule r must be generated by the intruder from his current knowledge. Condition (10) is similar to condition (4) in the ground model, where the substitution is now applied also to the set of positive facts in the state (i.e. instead of matching we now perform unification). Condition (11) states that the inequalities are conjoined with the conjunction of all formulae that $\mathit{subCont}(N_1\sigma, P\sigma)$ yields. For a set of negative facts N and a set of positive facts P, $\mathit{subCont}(N, P)$ generates a disjunction of inequalities that excludes all unifiers between two positive facts t and t' such that $\mathsf{not}(t) \in N$ and $t' \in P$. Note that in the special case that $t = t'$ we obtain the solution $\sigma = []$, and naturally we define $\bigvee_{i=1}^{0}\ \phi$ to be simply false for any ϕ. Finally, condition (12) describes the positive facts P' in the successor state, which result by removing the positive LHS facts from P (under σ) and adding the RHS facts (under σ).

Definition 11. *We define the set of* reachable lazy states *associated to a protocol (I, R, G) as $\mathit{lreach}(I, R) = \bigcup_{n \in \mathbb{N}} \mathit{lsucc}_R^n(I, \emptyset, \emptyset)$.*

We also call $\mathit{lreach}(I, R)$ the *lazy intruder model* of the protocol (I, R, G), or *lazy model* for short.

As we show in §B, the lazy model is equivalent to the ground model, in the sense that they represent the same set of reachable states:

Lemma 2. *$\mathit{reach}(I, R) = \cup_{(P,C,N) \in \mathit{lreach}(I,R)} [\![(P, C, N)]\!]$ for every initial state I and every set R of rules of the form* (1)*.*

Recall that we have defined that a protocol is secure iff *goalcheck* (which represents the negation of the property required by the protocol, i.e. it represents the attacks on the protocol) is empty for all reachable

ground states. A similar check suffices in the lazy intruder model. We define the lazy goal-check for a lazy state $S = (P, C, N)$ and a goal state g as

$$\mathit{lgoalcheck}_g(P, C, N) = \{\sigma \mid g\sigma \subseteq P\sigma\}\,.$$

If *lgoalcheck* is not empty in a reachable lazy state S, then either S represents an attack or S is *unsatisfiable*, i.e. its semantics is empty:

Theorem 2. *A protocol (I, R, G) is* secure *iff $\sigma \in \mathit{lgoalcheck}_g(P, C, N)$ implies $[\![(P, C, N)\sigma]\!] = \emptyset$ for all $(P, C, N) \in \mathit{lreach}(I, R)$ and all $g \in G$.*

Using the above results, we will now show how we can build an effective semi-decision procedure for protocol insecurity based on the lazy intruder. (In the case of a bounded number of sessions, our procedure is actually a decision procedure.) To this end, we have to tackle three problems.

First, the *lstep* function yields in general infinitely many successors, as there can be infinitely many unifiers σ for the positive facts of the rules and the current state. However, as we follow the free algebra assumption on the message terms, two unifiable terms always have a unique *mgu*, and we can, without loss of generality, focus on that unifier. (Note also that there are always finitely many *mgu*'s as the set of rules is finite and a lazy state contains finitely many facts.)

Second, we must represent the reachable states. The lazy infinite-state approach (cf. §3) provides a straightforward solution to this problem, where we represent the reachable states as the tree induced by the lazy intruder successor function. (In the case of an unbounded number of sessions, this tree is infinitely deep.) We can apply the lazy goal-check as a filter on this tree, to obtain the lazy goal states.

Third, we must check whether one of these lazy goal states is satisfiable, i.e. represents a possible attack. (We will see that also this check can be applied as a filter on the tree.) The constraint reduction is the key to achieve this task. By Theorem 1, we know that, for a well-formed constraint set C, the reduction produces a set of simple constraint sets that together have the same semantics as C. The following lemma shows that a lazy state with a simple constraint set and a satisfiable collection of inequalities is always satisfiable; note that the proof shows the key idea behind inequalities in our lazy intruder model, and also illustrates the advantages of our model with respect to other symbolic approaches.

Lemma 3. *Let (P, C, N) be a lazy state where C is simple and N is satisfiable (i.e. $\exists\sigma.\ \sigma \models N$). Then $[\![(P, C, N)]\!] \neq \emptyset$.*

Proof. In a simple constraint set, the messages the intruder has to generate consist only of uninstantiated variables, leaving his choice of messages for these variables open. Let us assume, as is standard, that the initial intruder knowledge is not empty, but that he knows at least his own name i. Hence, as the intruder can always generate some message from his knowledge, a simple constraint set is always satisfiable, i.e. $[\![C]\!] \neq \emptyset$ for a simple C. So far this is the same argumentation as in other constraint-based approaches like [27].

The key idea behind the integration of inequalities is that these cannot destroy the satisfiability of the constraint set, unless the inequalities alone are already unsatisfiable, which we now show. It is straightforward to check whether a conjunction of disjunctions of inequalities N is satisfiable: let $\{v_1, \ldots, v_n\} = \mathrm{vars}(N)$ and $\sigma = [v_1 \mapsto m_1, \ldots, v_n \mapsto m_n]$ for ground messages m_i with $m_i \neq m_j$ for all $1 \leq i, j \leq n$ with $i \neq j$; the resulting ground collection of inequalities $N' = N\sigma$ is satisfiable iff N is satisfiable. (And it is simple to check if N' is satisfiable as it is ground.) Hence, if N is satisfiable and C is a simple constraint set, then every solution $\sigma \in [\![C]\!]$ (extended to variables in N that do not occur in C) is also a solution for N, if it only maps every variable in N to different messages. So, to satisfy both C and N, it is sufficient that the intruder is able to generate finitely many different messages, i.e. the messages $m_1, \ldots, m_n$ above.

Since our model is untyped, the intruder can achieve this easily, for instance by composing in different ways the terms that he knows, even when he knows only his name i. That is, for $\{v_1, \ldots, v_n\} = \mathrm{vars}(N)$ a solution would be $\sigma = [v_1 \mapsto \mathsf{i}, v_2 \mapsto \langle \mathsf{i},\mathsf{i}\rangle, \ldots, v_n \mapsto \langle \mathsf{i}, \ldots\rangle]$. Now $\sigma \in [\![C]\!]$ and $\sigma \models N$, hence $P\sigma \in [\![(P, C, N)]\!]$ by definition, which concludes the proof. □

From this lemma we can conclude the following for a well-formed constraint set C and a collection of inequalities N. If there is at least one solution $(C', \tau) \in \mathit{Red}(C)$ and $N\tau$ is satisfiable, then $[\![(P, N, C)]\!] \neq \emptyset$,

since C' is simple and $[\![C']\!] \subseteq [\![C]\!]$, by Theorem 1. Otherwise, if $Red(C) = \emptyset$ or if N is unsatisfiable, then $[\![(P, C, N)]\!] = \emptyset$, also by Theorem 1.

So, for a reachable lazy state (P, C, N) we can decide if $[\![(P, C, N)]\!]$ is empty, as long as C is well-formed. To obtain simple constraint sets, we will call *Red*, which only applies to well-formed constraint sets. It thus remains to show that all constraint sets of reachable lazy states are well-formed, which follows from the way new constraints are generated during the *lstep* transitions.

Lemma 4. *For a protocol (I, R, G), if $(P, C, N) \in lreach_R(I)$ then C is well-formed.*

We have now put all pieces together to obtain an effective procedure for checking whether a protocol is secure: we generate reachable lazy states and filter them both for goal states and for constraint satisfiability. We will now briefly discuss how to implement this procedure in an efficient way.

4.5 Organizing State Exploration and Constraint Reduction

When implementing the lazy intruder we are faced with two design decisions: (1) in which order the two "filters" mentioned above are applied, and (2) how constraint reduction should be realized.

With respect to (1), note that the definition of reachable lazy states does not prescribe when *Red* should be called; *Red* is only used to determine if a constraint set is satisfiable. In OFMC we apply *Red* after each transition to check if the constraints are still satisfiable. This allows us to eliminate from the search all states with unsatisfiable constraint sets, as the successors of such states have again an unsatisfiable constraint set. We also extend this idea to checking the inequalities, and remove states with unsatisfiable inequalities. In the lazy infinite-state approach this can be simply realized by swapping the order in which the "filters" are applied, i.e. the tree of reachable lazy states is first filtered for satisfiable lazy states (using *Red*), thereby cutting several subtrees, and then for goal states (using *lgoalcheck*).

With respect to (2), note that the question of how to compute the constraint reduction (in particular, how to analyze the intruder knowledge) is often neglected in the presentations of other symbolic intruder approaches. The solution adopted in [12] is to proceed "on demand": a message in the intruder knowledge is analyzed iff the result of this analysis (possibly after the further analysis of its submessages) can be unified with a message the intruder has to generate. We adopt a more efficient solution. We apply the analysis rules to every constraint as long as they are applicable. The result is that the intruder knowledge is "normalized" with respect to the analysis rules. As a consequence, we need not consider analysis rules during the reduction of the constraints anymore. This has the advantage that to check if the G^l_{unif} rule is applicable to a message m that the intruder has to generate, we must simply check if in the (analyzed) intruder knowledge some message m' appears that can be unified with m. In contrast, with the analysis on demand it is in this case necessary to check if a unifiable message may be obtained through analysis.

However, when normalizing the intruder knowledge we must take care of the fact that the analysis may lead to substitutions. Every substitution restricts the set of possible solutions and in this case the restriction is only necessary if the respective decrypted content of the message is actually used later (which is, in contrast, elegantly handled by the analysis on demand).[5] Our solution to this problem is to distinguish between analysis steps that require a substitution and those that do not. The latter kind can be performed without restriction. The others are not performed; rather, we add to the intruder knowledge the term that would be obtained in case of a successful analysis (without unification), and *mark* the term to express that the intruder *may* know it, but only under a certain substitution. If a marked term is actually needed, then the respective analysis steps are performed. Otherwise, the marked term stays in the intruder knowledge.

Our strategy is more complex than the analysis on demand, but is more efficient in almost all cases, as the occurrence of marked terms is rare and the use of a marked term is even rarer. In the case where no marked term are used, the normalizing analysis is obviously preferable.

[5] As an example, suppose that the intruder wants to analyze the message $\{\!|\{\!|\mathsf{m}|\!\}_\mathsf{k}|\!\}_{\{|\mathsf{M}|\!\}_\mathsf{k}}$, where M is a variable symbol that represents a message the intruder has generated earlier, and that he already knows the message $\{\!|\mathsf{m}|\!\}_\mathsf{k}$. Obviously the new constraint expressing that the key-term can be derived from the rest of the knowledge, $\mathit{from}(\{\!|\mathsf{M}|\!\}_\mathsf{k}, \{\!|\mathsf{m}|\!\}_\mathsf{k})$, is satisfiable, unifying $\mathsf{M} = \mathsf{m}$. The point is that the result of the decryption does not give the intruder any new information (he already knows $\{\!|\mathsf{m}|\!\}_\mathsf{k}$), hence by unifying $\mathsf{M} = \mathsf{m}$ we limit unnecessarily the possible messages the intruder could have said.

Protocol Name	Kind of Attack	Time	Protocol Name	Kind of Attack	Time
ISO symm. key 1-pass unilateral auth.	Replay	0.0	*Kehne Langendorfer Schoenwalder (rep. part)*	Parallel-session	0.2
ISO symm. key 2-pass mutual auth.	Replay	0.0	*Kao Chow rep. auth., 1*	STS	0.5
Andrew Secure RPC prot.	Type flaw	0.0	*Kao Chow rep. auth., 2*	STS	0.5
	Replay	0.1	*Kao Chow rep. auth., 3*	STS	0.5
ISO CCF 1-pass unilateral auth.	Replay	0.0	*ISO public key 1-pass unilateral auth.*	Replay	0.0
ISO CCF 2-pass mutual auth.	Replay	0.0	*ISO public key 2-pass unilateral auth.*	Replay	0.0
Needham-Schroeder Conventional Key	STS	0.3	*Needham-Schroeder Public Key NSPK*	Man-in-the-middle	0.0
Denning-Sacco (symmetric)	Type flaw	0.0	*NSPK with key server*	Man-in-the-middle	1.1
Otway-Rees	Type flaw	0.0	*NSPK with Lowe's fix*	Type flaw	0.0
Wide Mouthed Frog	Parallel-session	0.2	*SPLICE/AS auth. prot.*	Replay	4.0
Yahalom	Type flaw	0.0	*Kwang and Chen's modified SPLICE*	Man-in-the-middle	0.0
Woo-Lam Π_1	Type flaw	0.0	*Denning Sacco Key Distr. with Public Key*	Man-in-the-middle	0.5
Woo-Lam Π_2	Type flaw	0.0	*Shamir Rivest Adelman Three Pass prot.*	Type flaw	0.0
Woo-Lam Π_3	Type flaw	0.0	*Encrypted Key Exchange*	Parallel-session	0.1
Woo-Lam Π	Parallel-session	0.2	*Davis Swick Private Key Certificates, prot. 1*	Type flaw	0.1
Woo-Lam Mutual auth.	Parallel-session	0.3		Replay	1.2
Needham-Schroeder Signature prot.	Man-in-the-middle	0.1	*Davis Swick Private Key Certificates, prot. 2*	Type flaw	0.2
* *Neuman Stubblebine initial part*	Type flaw	0.0		Replay	0.9
* *Neuman Stubblebine rep. part*	STS	0.0	*Davis Swick Private Key Certificates, prot. 3*	Replay	0.0
Neuman Stubblebine (complete)	Type flaw	0.0	*Davis Swick Private Key Certificates, prot. 4*	Replay	0.0

Table 1: Performance of OFMC over the Clark/Jacob library

5 Experimental Results

To assess the effectiveness and performance of our OFMC tool, we have tested it on a large protocol suite, which includes the authentication protocols of the Clark/Jacob library [13, 18], as well as a number of industrial-scale protocols. Since OFMC implements a semi-decision procedure, it does not terminate for correct protocols, even though it can establish the correctness of protocols for a bounded number of sessions. We describe below the search times we have measured for finding attacks in flawed protocols.

5.1 The Clark/Jacob Library

The OFMC can find an attack for 32 of the 33 flawed protocols of the Clark/Jacob library.[6] As the performance times in Table 1 show, our OFMC is a state-of-the-art tool: for each of the flawed protocols, a flaw is found in under 4 seconds and the total analysis of all flawed protocols takes less than one minute of CPU time. (Times are obtained on a PC with a 1.4GHz Pentium III processor and 512Mb of RAM, but note that, due to the use of iterative deepening search, the OFMC tool requires a negligible amount of memory.) To our knowledge, no other existing tool for finding attacks is this fast and has comparable coverage.

Note that the analysis of the untyped and typed IF specifications may lead to the detection of different kinds of attacks. When this is the case, in Table 1 we report the two attacks found. (In all other cases, the times are obtained using the default untyped model.) Also note that the table contains four variants of protocols in the library, marked with a "*", that we have additionally analyzed, and that "Replay STS" abbreviates replay attack based on a short-term secret. Table 1 also reports a new attack that we have found on the Yahalom protocol, which we describe in §A.3.

5.2 The H.530 Protocol

We have also applied OFMC to a number of industrial-scale protocols, such as IKE (for which we found the weaknesses already reported in [26]), and in particular, the H.530 protocol of the ITU [23], which was developed by Siemens. H.530 is a multimedia protocol that is deployed as shown in the left part of Fig. 3: a mobile terminal (MT) wants to establish a secure connection and negotiate a Diffie-Hellman key with the gatekeeper (VGK) of a visited domain. As they do not know each other in advance, the authentication is performed using an authentication facility AuF within the home domain of the MT; both MT and VGK initially have shared keys with AuF. The right part of Fig. 3 shows the messages exchanged: first, both MT and VGK create Diffie-Hellman half-keys, along with hashes that are encrypted for the AuF (denoted by the messages Req_{MT} and Req_{VGK}, respectively). After a successful check of these messages, AuF replies with appropriate acknowledge messages Ack_{MT} and Ack_{VGK} that also contain encrypted hashes for the respective

[6] The missing flaw is that of the CCITT X.509 protocol, which has the weakness that agents may sign messages they cannot analyze completely. The OFMC cannot find this attack since the HLPSL does not (yet) allow one to specify an appropriate goal that is violated by this weakness in the first place.

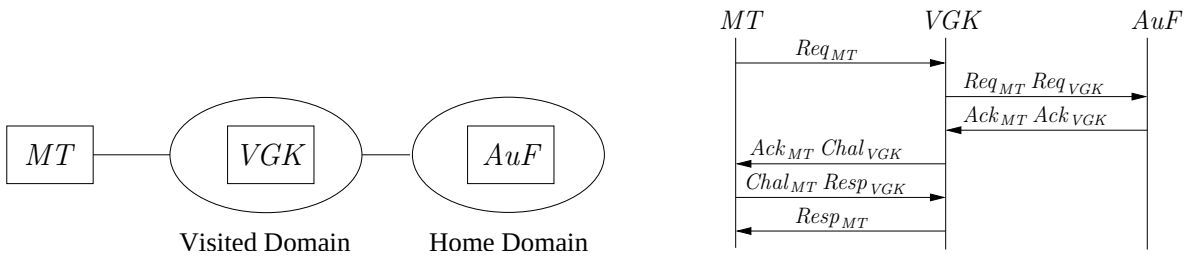

Figure 3: The H.530 protocol (simplified). The figure on the left shows the deployment of the protocol. On the right, the messages exchange between the participants is summarized.

recipients. Finally, MT and VGK perform a mutual challenge-response using the new Diffie-Hellman key that was authenticated by AuF (directly or over a chain of trustworthy servers).

The main problem that we had to tackle for the automated analysis of the H.530 protocol is the fact that the protocol employs the Diffie-Hellman key-agreement, which is based on a property of cryptographic algorithms (namely the commutativity of exponents) that violates the free algebra assumption. We lack space here to discuss in detail how we solved this problem in our model. The central point is this: while the messages exchanged in the H.530 protocol are considerably more complex than the ones of the Clark/Jacob protocols, this complexity is not a problem for our approach, unlike for other model-checking tools, e.g. [18, 25]. We could directly analyze the original specification of the H.530 without first simplifying the messages.

OFMC takes only 1.6 seconds to detect an unknown attack to H.530. It is a replay attack where the intruder first listens to a session between honest agents mt in role MT, vgk in role VGK, and auf in role AuF. Then the intruder starts a new session impersonating both mt and auf. The weakness that makes the replay possible is the lack of fresh information in the message Ack_{VGK}, i.e. the message where auf acknowledges to vgk that he is actually talking with mt. Replaying the respective message from the first session, the intruder impersonating mt can negotiate a new Diffie-Hellman key with vgk, "hijacking" mt's identity. To perform the attack, the intruder must at least be able to eavesdrop and insert messages both on the connection between MT and VGK, and on the connection between VGK and AuF. We have suggested including MT's Diffie-Hellman half-key in the encrypted hash of the message Ack_{VGK} to fix this problem. With this extension we have not found any further weaknesses of the protocol and Siemens has changed the protocol accordingly.

We conclude this section by noting that our tool achieves its performance by using the lazy intruder technique we have presented in the previous section; no further search heuristics or restrictions were applied. We believe that there is thus considerable potential for further improvements.

6 Related Work

Our formal protocol model is based on the specification languages HLPSL and IF, which we have been developing with colleagues as part of a larger project [2, 3, 11, 12, 32] aimed at providing security protocol validation tools. The HLPSL evolved out the Casrul system, described in [24].

The use of a generic high-level language and a lower-level language based on (multi)set rewriting was developed by [16] and our work was inspired by this combination. There are, however, a number of differences between their combination, CAPSL/CIL, and ours. For example, CAPSL cannot handle protocols where an agent first receives a message that he cannot decrypt, say $\{m\}_k$, and later receives the symmetric key k, which he can use to decrypt the message. In our case, the agent will store $\{m\}_k$ and later decrypt it after receiving the key. In comparison with CIL, the IF additionally supports negative facts, which extends the scope of protocols and properties that can be modeled. Finally, based on the available experiments (cf. §5 as well as [6, 16]), OFMC appears to be considerably more effective on the protocols we have analyzed than the current tools connected to CAPSL/CIL.

There are several model-checking approaches similar to ours. As a prominent example, we compare our approach with Casper [18, 25], a compiler that maps protocol specifications, written in a high-level language similar to CAPSL and HLPSL, into protocol descriptions in the process algebra CSP. The approach uses

finite-state model-checking with FDR2. Casper/FDR2 has successfully discovered flaws in a wide range of protocols: among the protocols of the Clark/Jacob library, it has found attacks on 20 protocols previously known to be insecure, as well as attacks on 10 other protocols originally reported as secure. Experiments indicate that OFMC is considerably faster than Casper/FDR2, despite being based on a more general model: Casper limits the size of messages to obtain a finite-state model. This limitation is problematic for the detection of type-flaw attacks, e.g. Casper/FDR also misses our type-flaw attack on Yahalom (cf. §A.3). Finally, Casper does not support non-atomic keys, which hinders its application to protocols like IKE, where each participant constructs only a part of the shared key that is negotiated.

The Athena tool [33] combines model-checking and interactive theorem-proving techniques with the strand-space model [20] to reduce the search space and automatically prove the correctness of security protocols with arbitrary numbers of concurrent runs. Interactive theorem-proving in this setting allows one to limit the search space drastically by manually proving lemmata (e.g. "the intruder cannot find out a certain private key, as it is never transmitted"). However, the amount of user interaction necessary to obtain such statements might be considerable. Moreover, like Casper/FDR2, Athena supports only atomic keys, and cannot detect type flaws.

We conclude by comparing our work with other related approaches to symbolically modeling intruder actions, expanding on the remarks in §4. The idea of a symbolic intruder model has undergone a steady evolution, becoming increasingly simpler and general. In the earliest work [22], both the technique itself and the proof were of substantial complexity. In [1, 8], both based on the spi-calculus, the technique and its formal presentation were considerably simplified. [27] generalized the system to support non-atomic symmetric keys and also introduced the use of constraint sets, which separate the underlying model and the symbolic representation. [15] improved the system of [27] by extending its expressiveness and providing a more efficient implementation.

In [11, 12], Chevalier and Vigneron lifted the restriction of a fixed public-key infrastructure, where every agent has a fixed key-pair and knows his own private key and each agent's public key. This work is the closest to ours. Both approaches are based on the IF and (multi)set rewriting (which is shown to be closely related to strands in [10]). However, there are important differences, as discussed in §4. For instance, as we explained in §4.2, we have removed all procedural aspects from the symbolic intruder rules, making the approach more declarative and the proofs simpler. Moreover, we have extended the lazy intruder by introducing inequalities, which, together with the notion of simple constraints, has a very natural interpretation: "the intruder can generate as many different terms as he likes".

As we have seen, most approaches are restricted to atomic keys.[7] This prevents modeling many modern protocols like IKE. Moreover, untyped protocol models with atomic keys exclude type-flaw attacks in which keys are confused with composed terms. We believe that this is the reason why our type-flaw attack on the Yahalom protocol was not discovered earlier, even though Yahalom has been extensively studied.

7 Conclusions and Future Work

We have presented an approach to security protocol analysis implemented by the model-checker OFMC, which represents a substantial development of the idea of lazy on-the-fly model-checking originally proposed in [5, 6]. The original tool required the use of heuristics and, even then, did not scale to most of the protocols in the Clark/Jacob library. The use of the symbolic techniques described here has made an improvement of many orders of magnitude and the techniques are so effective that heuristics play no role in the current system. Moreover, OFMC scales well beyond those protocols of Clark/Jacob, as our example of the H.530 illustrates.

Current work involves applying OFMC to other industrial-scale protocols, such as those proposed by the IETF. Although initial experience is positive, we see an eventual role for heuristics in leading to further improvements. For example, a simple evaluation function could be: "has the intruder learned anything new through this step, and how interesting is what he learned?" We have also been investigating the integration

[7] For example, [9] generalizes previous work [8] by introducing a generic set of cryptographic primitives, but the approach is still limited to atomic keys and it is unclear how that can be lifted without losing the genericity. Note that all cryptographic primitives that are given as examples in [9] are also implemented in OFMC.

of partial-order reduction techniques in our model-checker. The first results are very positive and we will report on them in a forthcoming paper [7].

References

1. R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. INRIA Research Report 3915, 2000.
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 *Proceedings of CAV'02*, LNCS 2404, pages 349–354. Springer-Verlag, 2002. URL of the AVISS and AVISPA projects: `www.avispa-project.org`.
3. A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *Proceedings of FORTE 2002*, LNCS 2529, pages 210–225. Springer-Verlag, 2002.
4. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
5. D. Basin. Lazy infinite-state analysis of security protocols. In *Proceedings of CQRE'99*, LNCS 1740, pages 30–42. Springer-Verlag, 1999.
6. D. Basin and G. Denker. Maude versus Haskell: an Experimental Comparison in Security Protocol Analysis. In *ENTCS 36*. Elsevier, 2001.
7. D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. Technical Report 405, Dep. of Computer Science, ETH Zurich, 2003. Available at `www.inf.ethz.ch/research/publications/`.
8. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP'01*, LNCS 2076, pages 667–681. Springer-Verlag, 2001.
9. M. Boreale and M. G. Buscemi. A framework for the analysis of security protocols. In *Proceedings of CONCUR 2002*, LNCS 2421, pages 483–498. Springer-Verlag, 2002.
10. I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *Proceedings of CSFW'00*, pages 35–51. IEEE Computer Society Press, 2000.
11. Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proceedings of ASE'01*. IEEE Computer Society Press, 2001.
12. Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In E. Brinksma and K. G. Larsen, editors, *Proceedings of CAV'02*, LNCS 2404, pages 324–337. Springer-Verlag, 2002.
13. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL: `www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.
14. E. Cohen. TAPS: A first-order verifier for cryptographic protocols. In *Proceedings of CSFW'00*. IEEE Computer Society Press, 2000.
15. R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proceedings of SAS 2002*, LNCS 2477, pages 326–341. Springer-Verlag, 2002.
16. G. Denker, J. Millen, and H. Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, October 2000. See `www.csl.sri.com/~millen/capsl/`.
17. D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
18. B. Donovan, P. Norris, and G. Lowe. Analyzing a Library of Security Protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.
19. N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.
20. F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.
21. M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proceedings of CSFW'01*. IEEE Computer Society Press, 2001.
22. A. Huima. Efficient infinite-state analysis of security protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

23. ITU-T Recommendation H.530: Symmetric Security Procedures for H.510 (Mobility for H.323 Multimedia Systems and Services). 2002.

24. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proceedings of LPAR 2000*, LNCS 1955, pages 131–160. Springer-Verlag, 2000.

25. G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998. See `http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/`.

26. C. Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1999.

27. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.

28. J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 141–153, 1997.

29. L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

30. L. C. Paulson. Relations between secrets: The Yahalom protocol. In *Proceedings of the 7th Cambridge International Workshop on Security Protocols*, LNCS 1796, pages 73–77. Springer-Verlag, 1999.

31. A. Perrig and D. Song. Looking for diamonds in the desert (extending automatic protocol generation to three-party authentication and key agreement protocols). In *Proceedings of CSFW'00*. IEEE Computer Society Press, 2000.

32. M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2001.

33. D. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9:47–74, 2001.

A The Yahalom Protocol

A.1 A HLPSL Specification

Figure 4 shows, on the left, the full HLPSL-specification of the Yahalom protocol, which aims at distributing a session key K_{AB} to agents playing in the roles A and B with the help of a trusted server playing in the role S. On the right is the trace of a new attack that we have found, as output by OFMC (cf. §A.3). As we remarked, the core of the specification is the list of messages exchanged between the agents acting in the protocol roles. We also specify additional information that is often left implicit (or that is explained informally) in protocol declarations. In the identifiers section, for instance, we specify the types of the identifiers used, which determines their properties. For example, here we declare symmetric keys and nonces.

Some further remarks on HLPSL are in order. First, we denote the encryption of a message M by the key K by writing $\{M\}_K$ (or `{M}K` in the ASCII input and output of our tool OFMC). Second, in the knowledge section, one specifies which atomic messages an agent playing a role of the protocol must initially have in order to execute the protocol in that role. All atomic messages that are not part of this initial knowledge, e.g. the nonces N_A and N_B in the Yahalom example, are *fresh*, i.e. they are created during the protocol execution by the agent that first uses them.

So far, the protocol description is generic, i.e. it specifies how an agent playing in a role of the protocol should behave. Every honest agent is a process that can participate in an unbounded number of parallel *sessions*, i.e. executions of the protocol, playing in any of the roles. To constrain search, we can bound this infinite set of possible protocol instantiations by specifying a finite set of agents who are prepared to play in a specified scenario. Note that the specification of such protocol scenarios does not result in a finite model, as an agent who has finished a session can always start a new one using fresh items.

For instance, the `Session_instances` section of the Yahalom example specifies that the agents named a and b wish to execute the protocol playing in the roles A and B respectively, and that a is also willing to talk to i, which is the HLPSL keyword for the intruder. Note that the intruder cannot only pose as any other agent, but he can also participate in a session as a normal agent under his real name. As we will see below, this particular scenario gives rise to a type-flaw attack on the Yahalom protocol. Note also that we

```
Protocol Yahalom;
Identifiers
  A,B,S: role;
  KAS,KBS,KAB: symmetric_key;
  NA,NB: nonce;
Knowledge
  A: B,S,KAB;
  B: A,S,KBS;
  S: A,B,KAS,KBS;
Messages
  1. A -> B: A,NA
  2. B -> S: B,{A,NA,NB}KBS
  3. S -> A: {B,KAB,NA,NB}KAS,{A,KAB}KBS
  4. A -> B: {A,KAB}KBS,{NB}KAB
Session_instances
  [A:a; B:b; S:s; KAS:kas; KBS:kbs]
  [A:i; B:b; S:s; KAS:kis; KBS:kbs];
Intruder divert, forge;
Intruder_knowledge A,B,S;
Goal B authenticate S on KAB;
```

```
Protocol Yahalom;
Statistics Time: 0.02 sec
Violated_goal: B authenticate S on KAB
Attack_trace
 1. i    -> b    : i,NA
 2. b    -> i(s) : b,{i,NA,fresh(idNB,2)}kbs
 2. i(b) -> s    : b,{i,NA,fresh(idNB,2)}kbs
 3. s    -> i    : {b,kab,NA,fresh(idNB,2)}kis,
                    {i,kab}kbs
 4. i    -> b    : {i,NA,fresh(idNB,2)}kbs,
                    {fresh(idNB,2)}(NA,fresh(idNB,2))
```

Figure 4: A HLPSL-specification of the Yahalom protocol and the corresponding output of the OFMC tool.

have recently implemented a new version of our tool where session instances are automatically generated and need not be specified explicitly.

Finally, we specify the security goal(s) that should be achieved by the protocol, which determines what constitutes an attack. Currently, the HLPSL supports *secrecy* and *authentication* goals. Secrecy of an atomic message, e.g. the nonce N_A or N_B in the Yahalom protocol, means that the intruder should not get hold of that message. Authentication is more complex: `B authenticate A on M` means that if an agent b playing in the role B has executed his part of a session, then the agent he believes to play in the role A has really sent to him the value he has accepted for M, and this value is not replayed, i.e. b has never accepted the same value before.

A.2 An Example of an IF Rule

To illustrate the output of the HLPSL2IF translator, we return to the Yahalom protocol and consider the following rule, where we omit the pairing operators to simplify the notation:

$$\mathsf{state}(3, \mathsf{A}, \mathsf{B}, \mathsf{S}, \mathsf{K_{AS}}, \mathsf{N_A}).\mathsf{message}(\{|\mathsf{B}, \mathsf{K_{AB}}, \mathsf{N_A}, \mathsf{N_B}|\}_{\mathsf{K_{AS}}}, \mathsf{K_{BS}_part})$$
$$\Rightarrow \mathsf{message}(\mathsf{K_{BS}_part}, \{|\mathsf{N_B}|\}_{\mathsf{K_{AB}}}).\mathsf{state}(\mathsf{final}, \mathsf{A}, \mathsf{B}, \mathsf{S}, \mathsf{K_{AS}}, \mathsf{N_A}, \mathsf{N_B}, \mathsf{K_{AB}})$$

This rule describes the behavior of an agent in role A who has sent message 1 of the protocol using the value $\mathsf{N_A}$, and waits for message 3, which must have the following form: the first component must be encrypted with the shared key of A and S, while the second component $\mathsf{K_{BS}_part}$ can be any message since A does not know the shared key $\mathsf{K_{BS}}$ and therefore cannot check this message. The agent, however, checks that the first part contains the nonce value $\mathsf{N_A}$ he originally sent and extracts from this message the values he uses in this protocol run for $\mathsf{N_B}$ and $\mathsf{K_{AB}}$. If such a message is received, the agent sends message 4 as an answer and goes to the final state of the protocol execution.

A.3 A New Attack

Figure 4 (on the right) presents a new attack that we have found on the Yahalom protocol. The Clark/Jacob library reports an attack on the protocol, but this attack requires that the intruder can guess the nonce N_B, which is contrary to the usual assumption of unguessability of nonces. Using OFMC, however, we have

uncovered a subtle weakness, which we display on the right-hand side of Figure 4. In this trace, the intruder officially (under his real name i) plays in the role A, and the agents b and s play in the roles B and S. The notation i(b) denotes that the intruder poses as b. According to the protocol, the intruder receives the new session-key kab from the server in message 3, along with a message encrypted for b, which should be forwarded to him in message 4. However, the intruder replays the encrypted part of message 2 instead. This is accepted by b since the message is encrypted with the expected key kbs and starts with the expected agent name i. Hence, b accepts the pairing of nonces NA, fresh(idNB,2) as the session-key issued by the server. Although the intruder doesn't "get in" with this attack (e.g. he did not make the agent b believe he talks with somebody else or find out a secret of other sessions), this state violates an authenticate goal, namely that any agent playing in the role B can rely on the integrity of the session-key: the intruder can make the agent b accept a fake key that was not originated by the server. A part of that key, i's nonce NA, is completely determined by the intruder.

We conclude with three remarks First, in [30] Paulson proved the security of the Yahalom protocol (including the goal we have found to be violated), but he used a typed model and the above attack exploits a type-confusion (between a key and a pair of nonces). Second, the attack trace given above is similar to the attack originally described in [13]. There, the intruder listens to the communication between honest agents, and then, similar to our attack, tries to generate message 4 abusing the same confusion with the encrypted part of message 2 as in our attack (which is, however, impossible unless the intruder can guess the nonce N_B). Finally, the attack we have detected is different from the one of [31], in which the intruder only makes the agent playing in the role B accept for the second time the key K_{AB} generated by the server. This is a replay attack. In our attack, the intruder makes the agent playing in the role B accept a key *different* from the one issued by the server.

B Proofs of the Lemmata and Theorems

We start by recalling that from the definition in §2 it follows that for every protocol description all reachable states are ground states, and that the intruder knowledge is monotonic.

Lemma 1 (Termination of *Red*). *$Red(C)$ is finite and recursively computable for every constraint set C.*

The intuition for the proof is that there can only be finitely many applications of the G^l_{unif} rule in any derivation and all other rules can be only applied finitely often before a unification operation is required. Like the other rules, the unification rule G^l_{unif} cannot introduce new variables. Either it is applied to ground terms or, if the terms to be unified are not ground, then it actually reduces the set of variables appearing in the constraints. The latter case cannot occur in a derivation an infinite number of times as there are only finitely many variables in a finite constraint set. The case of ground unifications as well as all other rule applications that can occur between two such substitutions is also limited: G^l_{unif} always reduces the LHS terms and G^l_{Dec} decomposes LHS terms. The applicability of analysis rules is bounded by the fact that at most all of the subterms of the initial RHS terms can be obtained. Formally:

Proof. We define a weight function w for messages on the LHS of a constraint as:

$$\begin{aligned} w(M) &= 1, \quad \text{for } M \in \mathit{AtomicMsg} \\ w(\{\!|M|\!\}_K) &= w(K) + w(M) + 1 \\ w(\langle M_1, M_2 \rangle) &= w(M_1) + w(M_2) + 1 \\ w(\{M_1, \ldots, M_n\}) &= \sum_{i=1}^{n} w(M_i) \end{aligned}$$

Note that here and below we focus on our restricted language and omit the cases for asymmetric encryption and inverse, which are similar to the ones presented.

The definition of the weight of the intruder knowledge, the RHS of the constraints, must take into account that analysis is possible, which introduces a new term into the intruder knowledge as well as a new constraint

for the derivation of the key. We define the weight function w_{IK} for messages in the IK on the RHS of a constraints as:

$$\begin{aligned}
w_{IK}(M) &= w(M), \quad \text{for } M \in \mathit{AtomicMsg} \\
w_{IK}(\{\!|M|\!\}_K) &= \begin{cases} w(\{\!|M|\!\}_K), \text{ if } M \in IK \\ w(\{\!|M|\!\}_K) + w_{M \cup IK}(M) + w(K) + w_{IK'}(IK') + 1, \text{ for } IK' = IK \setminus \{\!|M|\!\}_K, \text{ if } M \notin IK \end{cases} \\
w_{IK}(\langle M_1, M_2 \rangle) &= \begin{cases} w(\langle M_1, M_2 \rangle), \text{ if } M_1 \in IK \text{ and } M_2 \in IK \\ w(\langle M_1, M_2 \rangle) + w_{M_1 \cup M_2 \cup IK}(M_1 \cup M_2) + 1, \text{ if } M_1 \notin IK \text{ or } M_2 \notin IK \end{cases} \\
w_{IK}(\{M_1, \ldots, M_n\}) &= \sum_{i=1}^{n} w_{IK}(M_i)
\end{aligned}$$

We first show that if $IK' \subseteq IK$ then $w_{IK'}(M) < w_{IK}(M)$ for every set of messages M. In the recursive definition of w_{IK}, the only case where the indexed intruder knowledge matters is the case split for the encrypted and the paired messages. Clearly a smaller intruder knowledge leads to a greater weight in every case as the sum is larger when certain messages are not contained. By induction, we can conclude that this holds for all sets of messages M.

The weight of a constraint set $\{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_N, IK_n)\}$ is defined as a pair $W = (v, g)$, where v is the number of variables and g is sum of the weight of the constraints. That is:

$$W(\{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_N, IK_n)\}) = \left(\sum_{i=1}^{n} |\mathrm{vars}(T_i)| \, , \, \sum_{i=1}^{n} w(T_i) \; + \; w_{IK_i}(IK_i) \right)$$

We order these tuples lexicographically, overloading the "$<$" symbol, by defining:

$$(v_1, g_1) < (v_2, g_2) \text{ iff } v_1 < v_2 \text{ or } (v_1 = v_2 \text{ and } g_1 < g_2) \, .$$

This order is well-founded, as the ordering on both components is over the natural numbers. We show that the application of any constraint reduction rule decreases the weight of a constraint set according to $<$:

- G^l_{unif}: either no variable is substituted, then a term from the LHS is dropped (so strictly less in g), or at least one variable is substituted (so strictly less in v).
- G^l_{pair}, G^l_{scrypt}: strictly less in g since $1 + w(m_1) + w(m_2) > w(m_1) + w(m_2)$.
- A^l_{scrypt} (and similarly for A^l_{pair}): Consider the constraint set $\{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_n, IK_n)\}$. As the number of variables does not change, we consider the second component of the weight, which is

  $$\sum_{i=1}^{n} w(T_i) \; + \; w_{IK_i}(IK_i) \, .$$

  If we apply the analysis to a constraint number $k \in \{1, \ldots, n\}$ and $IK_k = \{t_1, \ldots, t_m\}$ and the analyzed message is $t_l = \{\!|M|\!\}_K$ for $l \in \{1, \ldots, m\}$, then the constraint after application of the analysis rule is

  $$\{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_{k-1}, IK_{k-1}), \; \mathit{from}(K, IK_k \backslash \{t_k\}), \mathit{from}(T_k, M \cup IK_k), \\ \mathit{from}(T_{k+1}, IK_{k+1}), \ldots, \mathit{from}(T_n, IK_n)\}$$

  with the weight (second component):

  $$\begin{aligned}
  & O + w(K) + w_{IK_0}(IK_0) + w(T_k) + w_{M \cup IK_k}(M \cup IK_k) \\
  = \quad & O + w(K) + w_{IK_0}(IK_0) + w(T_k) + w_{M \cup IK_k}(IK_0) + w(\{\!|M|\!\}_K) + w_{M \cup IK_k}(M) \\
  \leq \quad & O + w(K) + w_{IK_0}(IK_0) + w(T_k) + w_{IK_k}(IK_0) + w(\{\!|M|\!\}_K) + w_{M \cup IK_k}(M) \\
  =_{\mathrm{def}} \quad & O + w(T_k) + w_{IK_k}(IK_0) + w_{IK_k}(\{\!|M|\!\}_K) - 1 \\
  = \quad & O + w(T_k) + w_{IK_k}(IK_k) - 1 \\
  = \quad & \mathit{snd}(W(\{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_n, IK_n)\}) - 1
  \end{aligned}$$

  for $O = \mathit{snd}(W(\{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_n, IK_n)\} \setminus \mathit{from}(T_k, IK_k)))$ and $IK_0 = IK_k \setminus \{\!|M|\!\}_K$. So the analysis decreases the weight by at least one. This concludes the proof.

□

These results allow us to prove Theorem 1.

Theorem 1 (Correctness and completeness of *Red*). $[\![C]\!] = \{\sigma\sigma' \mid (C', \sigma) \in \mathit{Red}(C) \wedge \sigma' \in [\![C']\!]\}$ *for a well-formed constraint-set* C.

Proof. $\boxed{\supseteq}$: We begin with the "easy half" of the theorem, the correctness, meaning that the rules of the lazy intruder do not introduce new solutions to the semantics of the constraint set. It is enough to show that each rule application itself is correct. More precisely, we show that whenever the application of a rule transforms a constraint set and an initial substitution (C, τ) into $(C', \tau\sigma)$ (as noted at the beginning of §4.2, the new substitution can only be an extension of the old), then any solution for C' is also a solution for C, i.e. $\sigma' \in [\![C']\!] \implies \sigma\sigma' \in [\![C]\!]$. We show only the case of the rule G^l_{unif}, as the other cases are similar. In this case, C, C', and σ have the form: $C = \mathit{from}(t \cup T, s \cup IK) \cup C_0$, $\sigma = \mathit{mgu}(t, s)$, and $C' = (\mathit{from}(T, s \cup IK) \cup C_0)\sigma$. Let $\tau \in [\![C']\!]$. Then τ already satisfies all constraints in C except $\mathit{from}(t \cup T, s \cup IK)$. Now $t\tau\sigma' = s\tau\sigma'$ and hence $\tau\sigma'$ is a solution.

$\boxed{\subseteq}$: Showing the completeness of *Red*, i.e. that all solutions of a constraint set C are solutions of C's reduction, is more difficult. The main problem is that the completeness requires that *Red* is performed on a well-formed constraint set, but during the *Red* procedure the property (7) of the well-formedness can be destroyed by the analysis rules: an analysis rule introduces (i) a new constraint for the derivation of a key, where the intruder knowledge no longer contains the decryption key (hence the intruder knowledge may be smaller than the intruder knowledge of all previous constraints) and (ii) it adds the analyzed term to the intruder knowledge of the constraint to which it was applied (hence the intruder knowledge may be larger than the intruder knowledge in all successive constraints). Note that here and below we refer with previous and successive constraints to the order given by the well-formed constraints.

Problem (ii) can be easily overcome, since to restore property (7) we perform the same analysis steps also on the succesive constraints: these must allow the same derivations in the intruder knowledge as the initial constraint set was well-formed.

To tackle problem (i), we relax the invariant in the proof: at any step during the proof we want to preserve the invariant that the constraint set is well-formed if one removes all constraints that were introduced by an analysis rule. For simplicity we will still refer to this invariant as well-formedness. For the resulting simple constraints one can restore the well-formedness in the original sense by simply deleting the constraints that were introduced by analysis.[8]

We say in the following that *the result* (C', σ) *of a reduction or reduction step supports* τ if there exists σ' such that $\tau = \sigma\sigma'$ and $\sigma' \in [\![C']\!]$. Hence, our proof obligation is that for a given well-formed constraint C and an arbitrary solution $\tau \in [\![C]\!]$, the reduction yields at least one result $(C', \sigma) \in \mathit{Red}(C)$ that supports τ.

To show this, it is sufficient to prove that for a given well-formed, *non-simple* C and $\tau \in [\![C]\!]$ there exists at least one rule of the lazy intruder that can be applied to C such that the result of the rule application (C', σ) supports τ. Once this property of the rules is proven, we can show the completeness as follows. Let a well-formed constraint C and solution $\tau \in [\![C]\!]$ be given. We can repeatedly apply some of the lazy intruder rules, maintaining the solution τ, as long as the constraint set does not become simple anyway. By the termination lemma we know that an infinite chain of rule applications is impossible, so we must eventually reach a simple constraint set that supports τ.

So it remains to show that for a well-formed, non-simple constraint set C and a solution $\tau \in [\![C]\!]$, we can find a rule application such that the result supports τ. First, we introduce the notion of a *derivation tree*. Since $\tau \in [\![C]\!]$ we know there must be a τ derivation for every constraint $\mathit{from}(T, IK) \in C$ in the sense that $T\tau \subseteq \mathcal{DY}(IK\tau)$. We want to make explicit this derivation in the constraint set C by labelling every term in T with a $\mathcal{DY}$*-derivation tree*: A $\mathcal{DY}$*-derivation tree* is a binary tree, where leaves are messages, and each node is an application of one of the $\mathcal{DY}$ rules. Every leaf and every node stands for a message, composed

[8] This does not change the semantics of the constraint set. The constraint set is simple, therefore all constraints have only variables on the LHS. Moreover, it is well-formed without the constraints C that were introduced by the analysis, so the constraints C can only have variables on their LHS that were introduced by previous constraints. As these previous constraints have a smaller intruder knowledge, they are more restrictive, hence C is already entailed by them.

or decomposed from the respective subtrees. Hence, if $m \in \mathcal{DY}(IK)$, then there is a derivation tree such that m is the derived message at the root node and all leaves are in IK. We say that a constraint set C *is labelled with $\mathcal{DY}$-derivation trees for a solution* $\tau \in [\![C]\!]$, if in every constraint $\mathit{from}(T, IK) \in C$, every term $t \in T$ is labelled with the $\mathcal{DY}$-derivation tree of $t\tau$ from terms in $IK\tau$. As a simple example, consider $C = \mathit{from}(\langle K,m\rangle, \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})$, which has, among others, the solution $\tau = [K \mapsto \{\!|m|\!\}_k]$. We can label the message $\langle K,m\rangle$ with the derivation tree for the message $\langle K,m\rangle\tau = \langle\{\!|m|\!\}_k,m\rangle$ as follows:

$$\dfrac{\dfrac{\{\!|m|\!\}_k \in IK\tau}{\{\!|m|\!\}_k \in \mathcal{DY}(IK\tau)}\, G_{\text{axiom}} \quad \dfrac{\dfrac{\dfrac{\{\!|m|\!\}_k \in IK\tau}{\{\!|m|\!\}_k \in \mathcal{DY}(IK\tau)}\, G_{\text{axiom}} \quad \dfrac{\{\!|k|\!\}_{\{\!|m|\!\}_k} \in IK\tau}{\{\!|k|\!\}_{\{\!|m|\!\}_k} \in \mathcal{DY}(IK\tau)}\, G_{\text{axiom}}}{k \in \mathcal{DY}(IK\tau)}\, A_{\text{scrypt}} \quad \dfrac{\{\!|m|\!\}_k \in IK\tau}{\{\!|m|\!\}_k \in \mathcal{DY}(IK\tau)}\, G_{\text{axiom}}}{m \in \mathcal{DY}(IK\tau)}\, A_{\text{scrypt}}}{\langle\{\!|m|\!\}_k,m\rangle \in \mathcal{DY}(IK\tau)}\, G_{\text{pair}}}{\boxed{\langle K,m\rangle}}$$

where $IK\tau = \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k}$.

Let a well-formed, non-simple constraint set C and a solution $\tau \in [\![C]\!]$ be given, where C is labelled with $\mathcal{DY}$-derivation trees for the solution τ. According to the order of the well-formed constraint, we pick the first constraint $\mathit{from}(T, IK)$ that contains a non-variable message $t \in T$. We show that, depending on the root node of the $\mathcal{DY}$-tree t is labelled with, we can find a rule of the lazy intruder that is applicable and such that the resulting constraint set C' can again be labelled with $\mathcal{DY}$ trees according to τ (and hence the result still supports τ). We make a case split over the kind of root node that the $\mathcal{DY}$-tree has for t:

G_{axiom}: This means $t\tau \in IK\tau$. So t can be unified with a term $s \in IK$ and the G^l_{unif} rule is applicable to C, and the unifier $\sigma = mgu(t, s)$ is compatible with τ. Hence the resulting C' supports τ as all remaining terms can be labelled with the same trees as in C.

G_{scrypt} (respectively G_{pair}): Since t is not a variable, it must have the form $t = \{\!|t_2|\!\}_{t_1}$ (respectively $t = \langle t_1,t_2\rangle$) for some terms t_1 and t_2. As a result, the rule G^l_{scrypt} (respectively G^l_{pair}) can be applied. The resulting constraint contains the terms t_1 and t_2 which can be labelled with the respective subtrees of the derivation tree of t. Hence C' still supports τ.

A_{scrypt} (respectively A_{pair}): $t\tau$ is obtained through a decryption of a term $\{\!|t\tau|\!\}_k \in IK\sigma$ (respectively decomposition of a term $\langle t\tau,t_2\rangle$ or $\langle t_2,t\tau\rangle$). The respective analysis step of the lazy intruder may not yet be possible, if the subtrees of t's derivation tree contain further analysis operations (that must be performed first). Since the tree is finite, there must be an analysis operation in the $\mathcal{DY}$-tree for t that has no further analysis operations in its subtrees; we pick this analysis operation for performing the next lazy intruder rule on the constraint set. Let t_a be the term obtained in the $\mathcal{DY}$-tree by the selected analysis operation, and k and $\{\!|t_a|\!\}_k$ (respectively $\langle t_a,t_2\rangle$ or $\langle t_2,t_a\rangle$) the terms obtained at the children nodes of the analysis operation. There must be a message $T_A \in IK$ such that $T_A\tau = \{\!|t_a|\!\}_k$ (respectively $T_A\tau = \langle t_a,t_2\rangle$ or $T_A\tau = \langle t_2,t_a\rangle$). Without loss of generality, we can assume T_A is not a variable: if it is, then since C is well-formed $T_A \in T$ for an earlier constraint $\mathit{from}(T, IK_0) \in C$ for some $IK_0 \subseteq IK$ with $T_A \notin IK_0$; therefore the term to be constructed can be generated from the rest of the knowledge. Since T_A is not a variable, it must have the form $T_A = \{\!|M|\!\}_K$ (respectively $T_A = \langle T_1,T_2\rangle$), and the rule A^l_{scrypt} (respectively A^l_{pair}) can be applied. This adds the analyzed term M (respectively terms T_1 and T_2) to the intruder knowledge and the new constraint $\mathit{from}(K, IK \setminus \{\!|M|\!\}_K)$ to the constraint set (in the case of A_{pair}, no new constraint is added). Now all occurrences of the analyzed term t_a in the $\mathcal{DY}$-tree can be replaced with a leaf-node; this labelling is correct, as $M\tau = t_a$ (respectively $T_1\tau = t_a$ or $T_2\tau = t_a$). In the newly added constraint $\mathit{from}(K, IK \setminus \{\!|M|\!\}_K)$ (in the case of A_{scrypt}) the term K can be labelled with the k-subtree of the analysis node in the $\mathcal{DY}$-tree. This labelling is correct since this subtree cannot contain any further analysis operations; therefore it can contain only subterms of k (if k is non-atomic) that are present in $IK\tau$. In particular it cannot contain $\{\!|M|\!\}_K\tau = \{\!|t_a|\!\}_k$. Hence C' still supports τ.

We consider again the example constraint presented above and show how the reduction proceeds according to the previous case split. First, the root of the only term to generate is G_{pair}. Therefore we can apply the G^l_{pair} to obtain $\mathit{from}(K \cup m, \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})$ labelled as follows (recall we consider the solution $\tau = [K \mapsto \{\!|m|\!\}_k]$ and $IK\tau = \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k}$):

$$\frac{\{\!|m|\!\}_k \in IK\tau}{\{\!|m|\!\}_k \in \mathcal{DY}(IK\tau)}\ G_{\text{axiom}} \qquad \boxed{K}$$

$$\frac{\dfrac{\dfrac{\{\!|m|\!\}_k \in IK\tau}{\{\!|m|\!\}_k \in \mathcal{DY}(IK\tau)}\ G_{\text{axiom}} \quad \dfrac{\{\!|k|\!\}_{\{\!|m|\!\}_k} \in IK\tau}{\{\!|k|\!\}_{\{\!|m|\!\}_k} \in \mathcal{DY}(IK\tau)}\ G_{\text{axiom}}}{k \in \mathcal{DY}(IK\tau)}\ A_{\text{scrypt}} \quad \dfrac{\{\!|m|\!\}_k \in IK\tau}{\{\!|m|\!\}_k \in \mathcal{DY}(IK\tau)}\ G_{\text{axiom}}}{m \in \mathcal{DY}(IK\tau)}\ A_{\text{scrypt}} \qquad \boxed{m}$$

Since K is a variable in the constraint set, we can only proceed by deriving m. As the root of the derivation tree is an analysis operation and one subtree contains a further analysis step, we proceed with this innermost analysis. The respective analysis rule decrypts $\{\!|k|\!\}_{\{\!|m|\!\}_k}$, adds k to the intruder knowledge and the new constraint that the key term, $\{\!|m|\!\}_k$, can be derived from the rest of the knowledge, i.e. $\mathit{from}(\{\!|m|\!\}_k, \{\!|m|\!\}_k), \mathit{from}(K \cup m, k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})$, with the labelling:

$$\frac{\{\!|m|\!\}_k \in \{\{\!|m|\!\}_k\}}{\{\!|m|\!\}_k \in \mathcal{DY}(\{\!|m|\!\}_k)}\ G_{\text{axiom}} \qquad \boxed{\{\!|m|\!\}_k}$$

$$\frac{\{\!|m|\!\}_k \in (k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}{\{\!|m|\!\}_k \in \mathcal{DY}(k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}\ G_{\text{axiom}} \qquad \boxed{K}$$

$$\frac{\dfrac{k \in (k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}{k \in \mathcal{DY}(k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}\ G_{\text{axiom}} \quad \dfrac{\{\!|m|\!\}_k \in (k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}{\{\!|m|\!\}_k \in \mathcal{DY}(k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}\ G_{\text{axiom}}}{m \in \mathcal{DY}(k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}\ A_{\text{scrypt}} \qquad \boxed{m}$$

The first constraint is easily handled by the G^l_{unif} rule; we proceed then with the analysis of $\{\!|m|\!\}_k$, as the $\mathcal{DY}$-tree contains no further analysis operations. This introduces the new constraint for the derivation of k, i.e. $\mathit{from}(k, k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k}), \mathit{from}(K \cup m, m \cup k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})$, with the labelling:

$$\frac{k \in (k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}{k \in \mathcal{DY}(k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}\ G_{\text{axiom}} \qquad \boxed{k}$$

$$\frac{\{\!|m|\!\}_k \in (m \cup k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}{\{\!|m|\!\}_k \in \mathcal{DY}(m \cup k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}\ G_{\text{axiom}} \qquad \boxed{K}$$

$$\frac{m \in (m \cup k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}{m \in \mathcal{DY}(m \cup k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})}\ G_{\text{axiom}} \qquad \boxed{m}$$

Two further applications of G^l_{unif} then result into the simple constraint set that supports τ, i.e. $\mathit{from}(K, m \cup k \cup \{\!|m|\!\}_k \cup \{\!|k|\!\}_{\{\!|m|\!\}_k})$. This concludes the proof.

□

Lemma 2. *$\mathit{reach}(I, R) = \cup_{(P,C,N) \in \mathit{lreach}(I,R)} [\![(P, C, N)]\!]$ for every initial state I and every set R of rules of the form* (1)*.*

Proof. We show that a corresponding property holds for single transitions with *step* and *lstep* for every rule r of the form $\mathsf{message}(m_1).\mathsf{state}(m_2).P_1.N_1 \Rightarrow \mathsf{state}(m_3).\mathsf{message}(m_4).P_2$ and every lazy state (P, C, N):

$$\bigcup_{S \in [\![(P,C,N)]\!]} \mathit{step}_r S = \bigcup_{(P',C',N') \in \mathit{lstep}_r(P,C,N)} [\![(P', C', N')]\!] \tag{13}$$

where it must hold that $\mathrm{vars}(P, C, N) \cup \mathrm{vars}(r) = \emptyset$; this is ensured by the renaming performed as part of the *lsucc* function. The lemma follows by induction on the number of transitions.

$\boxed{(13)\ \subseteq}$: Given a lazy state (P, C, N) and ground states S and S', such that $S \in [\![(P, C, N)]\!]$ and $S' \in \mathit{step}_r(S)$. We show that there is a lazy state $(P', C', N') \in \mathit{lstep}_r S$ and $S' = [\![(P', C', N')]\!]$.

Since $S \in [\![(P, C, N)]\!]$ there is a substitution $\sigma \in [\![C]\!]$ with $\sigma \models N$ and $S = P\sigma$ and since $S' \in \mathit{step}_r S$, there is a substitution ρ such that

$$\begin{aligned}
&\mathrm{ground}(\rho) \ \wedge\ (\mathrm{dom}(\rho) = \mathrm{vars}(m_1, m_2, P_1)) \wedge \\
&m_1\rho \in \mathcal{DY}(\{\ i \mid \mathsf{i_knows}(i) \in S\}) \wedge \\
&\mathsf{state}(m_2\rho) \in S \ \wedge\ P_1\rho \subseteq S \wedge \\
&\forall f.\ \mathsf{not}(f) \in N_1 \implies f\rho \notin S \wedge \\
&S' = (S \setminus (\mathsf{state}(m_2\rho) \cup P_1\rho)) \cup \mathsf{state}(m_3\rho) \cup \mathsf{i_knows}(m_4\rho) \cup P_2\rho\,.
\end{aligned}$$

Given two such substitutions σ and ρ with the above properties, let $\tau = \sigma\rho$ and the lazy state (P', C', N') be defined as

$$\begin{aligned}
P' &= (P\tau \setminus (\mathsf{state}(m_2\tau) \cup P_1\tau)) \ \cup\ \mathsf{state}(m_3\tau) \cup \mathsf{i_knows}(m_4\tau) \cup P_2\tau \\
C' &= (C \cup \mathit{from}(m_1, \{i \mid \mathsf{i_knows}(i) \in P\}))\tau \\
N' &= N \wedge \bigwedge_{\phi \in \mathit{subCont}(N_1\tau, P\tau)} \phi\,.
\end{aligned}$$

Note that since $N_1\tau$ and $P\tau$ are ground, either $N' = N$ (if no term negated in $N_1\tau$ appears in $P\tau$) or $N' = N \wedge \mathit{false}$. Since $\forall f.\ \mathsf{not}(f) \in N_1 \implies f\rho \notin S$, $N' = N$ is the case.

Now it follows that $(P', C', N') \in \mathit{lstep}_r(P, C, N)$, $S' = P'\tau$ and it remains to show $S' \in [\![(P', C', N')]\!]$. Since $\sigma \in [\![C]\!]$ and $m_1\rho \in \mathcal{DY}(\{\ i \mid \mathsf{i_knows}(i) \in S\})$ it follows that $\tau \in [\![C']\!]$ and since $\sigma \models N$ and $N' = N$ we can conclude that $S' \in [\![(P', C', N')]\!]$, finishing the proof for the $\subseteq$ direction.

$\boxed{(13)\ \supseteq}$: This proof is similar, showing that for lazy states (P, C, N) and $(P', C', N') \in \mathit{lstep}(P, C, N)$ and a ground state $S' \in [\![(P', C', N')]\!]$, that there is a ground state $S \in [\![(P, C, N)]\!]$ with $S' \in [\![S]\!]$. □

Theorem 2. *A protocol (I, R, G) is* secure *iff $\sigma \in \mathit{lgoalcheck}_g(P, C, N)$ implies $[\![(P, C, N)\sigma]\!] = \emptyset$ for all $(P, C, N) \in \mathit{lreach}(I, R)$ and all $g \in G$.*

Proof. In this proof we need the following property:

$$\{S \mid \sigma \in \mathit{lgoalcheck}_G(P, N, R) \wedge S \in [\![(P, N, R)\sigma]\!]\} = \{S' \mid S \in [\![(P, N, R)]\!] \wedge \sigma \in \mathit{goalcheck}_G(S) \wedge S' = S\sigma\} \tag{14}$$

This proof is very similar to the proof property of (13) (in the proof of Theorem 2) and therefore we omit it here. Now by definition, (I, R, G) is secure iff $\forall S.\ S \in \mathit{reach}(I, R) \implies \mathit{goalcheck}_G(S) = \emptyset$. By Theorem 1, this is equivalent to the condition $\forall S.\ S \in \cup_{(P,C,N) \in \mathit{lreach}(I,R)} [\![(P, C, N)]\!] \implies \mathit{goalcheck}_G(S) = \emptyset$, which is equivalent to $\forall (P, C, N), S.\ (P, C, N) \in \mathit{lreach}(I, R) \ \wedge\ S \in [\![(P, C, N)]\!] \implies \mathit{goalcheck}_G(S) = \emptyset$. Finally, with property (14) this is equivalent to $\forall (P, C, N).\ (P, C, N) \in \mathit{lreach}(I, R) \ \wedge\ \sigma \in \mathit{lgoalcheck}_G(S) = \emptyset \implies [\![(P, N, C)\sigma]\!] = \emptyset$, which concludes the proof.

□

Lemma 4 *For a protocol (I, R, G), if $(P, C, N) \in \mathit{lreach}_R(I)$, then C is well-formed.*

Proof. We show the following stronger invariant of lsucc_R on lazy states (P, C, N) where all IF rules $r \in R$ have the form (1): C is well-formed and $\mathrm{vars}(P) \subseteq \mathrm{vars}(C)$. Consider the transition for one arbitrary $r \in R$. All variables that appear in r must appear in its positive LHS facts P_1. As we rename the lazy state before each transition, none of these variables occurs in the state. By the invariant, all variables in P appear in C as C is well-formed. All variables in r except those that only appear in the $\mathsf{message}$ facts are therefore substituted for a message term that contains only variables of the lazy state. As a result, the only new variables introduced by the transition are introduced on the LHS of a new constraint. The intruder knowledge of this constraint is a superset of the intruder knowledge in all previous constraints (this is because, by the form of the rules the intruder knowledge is monotonic, i.e. the intruder never forgets

messages). Also, the new constraint's intruder knowledge contains only variables that already appear in the state before the transition (and hence appear in some other constraints on the left side). Finally, the newly learned message can only contain variables that were introduced by the intruder-generated term or that were already present in the state before the transition. Therefore, since this reasoning applies to all constraints in the set, the new constraint set is well-formed. Moreover, as variables can only disappear from the constraint set after a substitution (or by applying *Red*), the facts P cannot contain variables that do not appear in C. Hence both invariants are preserved, and we conclude the proof of the lemma. □