ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Technical Report No. 450

OFMC: A Symbolic Model-Checker for Security Protocols*

David Basin Sebastian Mödersheim Luca Viganò

Department of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland
www.inf.ethz.ch/~{basin,moedersheim,vigano}
{basin,moedersheim,vigano}@inf.ethz.ch

June 11, 2004

Abstract

We present 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 very large, or even 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 known attacks and discovers a new one in a test-suite of 38 protocols from the Clark/Jacob library in a few seconds 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 developing efficient algorithms to automatically analyze properties of systems modeled as transition systems. A wide variety of model-checking approaches have been developed for analyzing security protocols, e.g. [1, 12, 28, 41, 43, 48, 49]. The key challenge they face is that the general security problem is undecidable [29], 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 time 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 [7, 8] of using *lazy data-types* to model the infinite state-space associated with a protocol. A lazy data-type is one where data-constructors (e.g. *cons* for building lists, or *node* for building trees) build data 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 [7, 8], lazy data-types are used to build, and compute with, models of security protocols: a protocol and a description of the

*This work was partially supported by the FET Open Project IST-2001-39252 and the BBW Project 02.0431, "AVISPA: Automated Validation of Internet Security Protocols and Applications". A preliminary version of this paper appeared in the proceedings of ESORICS 2003 [9]; this paper will appear in IJIS.

powers of an intruder are formalized as an infinite tree. Lazy evaluation is used to decouple the model from search and 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 does not address the problem of the prolific Dolev-Yao intruder and hence scales poorly. We show how to incorporate the use of symbolic techniques to substantially reduce this problem. We formalize a technique that significantly reduces the search space without excluding any attacks. This technique, which we call the *lazy intruder*, represents terms symbolically to avoid explicitly enumerating the possible messages the Dolev-Yao intruder can generate. This is achieved by representing intruder messages using terms with variables, and storing and manipulating constraints about what terms must be generated and which terms may be used to generate them.

The lazy intruder is a general, technology-independent technique that can be effectively incorporated in different approaches to protocol analysis. Here, we combine it 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 extend previous approaches, e.g. [1, 11, 12, 19, 25, 31, 32, 41], to symbolically representing the intruder and thereby extend the applicability of the lazy intruder technique to a larger class of protocols and properties. Second, despite the extensions, we simplify 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 formalize the integration of the technique into the search procedure induced by the rewriting approach of our underlying protocol model (this model provides an infinite-state transition system). Fourth, we also describe how to efficiently implement the lazy intruder, i.e. how to organize state exploration and constraint reduction. Finally, we present new ideas for organizing and controlling search based on searching different protocol scenarios, corresponding to different "sessions" where different agents assume different roles in the interleaved protocol executions. Our contribution here is to show how a technique that we call *symbolic session generation* can be used to exploit the symbolic representation of the lazy intruder and thereby avoid enumerating all possible session instances associated with a bounded number of sessions.

The result is OFMC, an on-the-fly model-checker for security protocol analysis. We have carried out a large number of experiments to validate our approach. For example, the OFMC tool finds all known attacks and discovers a new one (on the Yahalom protocol) in a test-suite of 38 protocols from the Clark/Jacob library [21] in a few seconds of CPU time for the entire suite. Moreover, we have successfully applied OFMC to a number of large-scale protocols including (subprotocols of) IKE, SET, and various other industrial protocols currently being standardized by the Internet Engineering Task Force (IETF). As an example of industrial-scale problem, we describe in this paper our analysis of the H.530 protocol [33], a protocol developed 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 revised the protocol [34].

Organization.

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 constraint reduction. We discuss the organization of state exploration and constraint reduction in §5 and present symbolic sessions in §6. We present experimental results in §7, and we discuss related and future work in §8. The appendix contains the proofs of the theorems and lemmata given in the body of the paper.

2 Protocol Specification Languages and Model

The formal model we use for protocol analysis is based on two specification languages: a high-level language (HLPSL) and a low-level one (IF). These languages have been developed in the context of the AVISPA project [5].

2.1 The High-Level Protocol Specification Language

The *High-Level Protocol Specification Language HLPSL* allows users to specify protocols in an Alice&Bob-style notation. As most of the ideas behind the HLPSL are standard, e.g. [26, 35], we explain its main features on an example. Fig. 1 shows the HLPSL-specification of the Yahalom protocol, which aims at distributing a session key `KAB` to two agents playing in the roles `A` and `B`; to do this, it uses a trusted server playing in the role `S`. The figure also contains the trace of a new attack that our tool OFMC has found, which we discuss in §7.2.

The core of the specification is the list of messages exchanged between the agents acting in the protocol roles. In the ASCII syntax of HLPSL, we denote the encryption of a message `M` with a symmetric key `K` by writing `{|M|}K` (and we write `{M}K` for the encryption of a message `M` with an asymmetric key `K`). HLPSL also allows one to specify information that is often left implicit (or that is explained informally) in protocol declarations. In the identifiers section, for instance, we declare the types of the identifiers used, which determines their properties. In the example, we declare a function `k` (representing a key-table), the new symmetric key `KAB`, and nonces `NA` and `NB` that are generated during protocol execution. Although not displayed in this example, HLPSL also supports asymmetric encryption, cryptographic hash-functions, non-atomic keys, and exponentiation.

In the knowledge section, one specifies which atomic messages an agent playing in a role of the protocol must initially have in order to execute the protocol in that role. For instance, an agent playing in the role `A` must initially know the names of the agents playing in the roles `B` and `S`, as well as the key `k(A,S)` he shares with `S`. All atomic messages that are not part of this initial knowledge, e.g. the nonces `NA` and `NB` 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* (or *session instances*), 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 *scenarios*, which are finite sets of sessions, i.e. instantiations of roles with agent names, where session numbers (IDs) are used to distinguish parallel sessions between the same agents.

For instance, the `Session_instances` section of the Yahalom example specifies two sessions: one where the agents named `a`, `b` and `s` execute the protocol playing in the roles `A`, `B` and `S` respectively, and one where the three roles are played by `i`, `b` and `s`, where `i` is the HLPSL keyword for the intruder. Note that the intruder can not 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 in §7.2, the particular scenario given in Fig. 1 is the one that gives rise to the new type-flaw attack on the Yahalom protocol found by OFMC. Note that the specification of such scenarios by the user is generally not desirable and, as we will show in §6, symbolic session generation exploits the symbolic representation of the lazy intruder to avoid enumerating session instances.

Finally, we specify the initial knowledge of the intruder and the security goal(s) that should be achieved by the protocol, which determines what constitutes an attack. Currently, HLPSL supports different forms of *authentication* and *secrecy* goals. Secrecy of an atomic message, e.g. the nonce `NA` or `NB` in the Yahalom protocol, means that the intruder should not get hold of that message (unless he is explicitly allowed to). Authentication is more complex: `B authenticates 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 that he has accepted for `M`, and this value is not replayed, i.e. `b` has never accepted the same value before.

HLPSL-specification:

```
Protocol Yahalom;
  Identifiers
    A,B,S: role;
    k: function;
    KAB: symmetric_key;
    NA,NB: nonce;
  Knowledge
    A: B,S,k(A,S);
    B: A,S,k(B,S);
    S: A,B,k;
  Messages
    1. A -> B: A,NA
    2. B -> S: B,{|A,NA,NB|}k(B,S)
    3. S -> A: {|B,KAB,NA,NB|}k(A,S),{|A,KAB|}k(B,S)
    4. A -> B: {|A,KAB|}k(B,S),{|NB|}KAB
  Session_instances
    [A:a; B:b; S:s]
    [A:i; B:b; S:s];
  Intruder_knowledge A,B,S;
  Goal B authenticates S on KAB
```

Output of OFMC:

```
Protocol Yahalom;
  Time: 0.02 sec
  Violated_goal: B authenticates S on KAB
  Attack_trace
    1. i    -> b    : i,NA
    2. b    -> i(s) : b,{|i,NA,fresh(idNB,sess2)|}k(b,s)
    2. i(b) -> s    : b,{|i,NA,fresh(idNB,sess2)|}k(b,s)
    3. s    -> i    : {|b,fresh(idKAB,sess2),NA,
                       fresh(idNB,sess2)|}k(i,s),
                       {|i,fresh(idKAB,sess2)|}k(b,s)
    4. i    -> b    : {|i,NA,fresh(idNB,sess2)|}kbs,
                       {|fresh(idNB,sess2)|}(NA,fresh(idNB,sess2))
```

Figure 1: A HLPSL-specification of the Yahalom protocol and OFMC's output.

A translator called HLPSL2IF (which has also been developed in the context of the AVISPA project) automatically translates a high-level HLPSL-specification into a low-level *Intermediate Format IF* based on first-order set rewriting. The IF is a simple but expressive formalism that is well-suited for the automated analysis of security protocols. OFMC takes IF specifications as input and we hence base our presentation on IF for concreteness, but the approach and methods we present in this paper can be applied to other kinds of protocol models like strand spaces or process calculi [30, 38, 48].

2.2 The Syntax of the Intermediate Format

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:*

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

We write $\mathcal{L}(n)$ for the context-free language associated to the non-terminal n. We write vars(t) *to denote the set of variables occurring in a (message, fact, or state)* term t*, and when* vars$(t) = \emptyset$ *we say that t is* ground *and write* ground(t)*. We straightforwardly extend the functions* vars *and* ground *to the more complex terms and structures defined below.*

Notation 1. We employ the following notation: we denote IF constants with lower-case sans-serif, IF variables with upper-case sans-serif, meta-variables (i.e. variables ranging over message terms) with *lower-case italics*, and sets with *upper-case italics*. □

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 that c is intended for. For instance, returning to the example in Fig. 1, the constant sess2 in the fresh terms fresh(idNB,sess2) and fresh(idKAB,sess2) indicates that the honest agents who created them are those declared in the second session instance (cf. also §7.2).

Messages in the IF are atomic messages or are composed using *pairing* $\langle m_1,m_2 \rangle$, or the *cryptographic operators* $\{m_2\}_{m_1}$ and $\{|m_2|\}_{m_1}$ (for *asymmetric* and *symmetric encryption* of m_2 with m_1), or $f(m)$ (for application of the function f to the message m, representing a hash-function or key-table), or m^{-1} (the *asymmetric inverse* of m).[1] Note that by default the IF is untyped (and the complexity of messages is not bounded), but it can also be generated in a typed variant. The typed variant leads to smaller search spaces at the cost of abstracting away possible type-flaw attacks on protocols.

[1] Some approaches, e.g. [44], denote by k^{-1} the inverse of a symmetric key k, with $k^{-1} = k$. We cannot do this since in our model messages are untyped and hence the inverse key cannot be determined from the (type of the) key. In our model, every message has an asymmetric inverse. As we will define, cf. Definition 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 key is to know it initially, to receive it in a message, or when it is the private key of a self-generated asymmetric key-pair.

Note also that we follow the standard *perfect cryptography assumption* that 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.[2] OFMC provides preliminary support for algebraic properties of operators like exponentiation, used for instance to model Diffie-Hellman key-exchange. Principled techniques exist for incorporating equational operator specifications into search, e.g. [13, 16, 17, 24, 42]; the description of the integration of such techniques is, however, outside the scope of this paper.

Note too that unlike other models, e.g. [27, 41], 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 specify protocols where keys are generated, distributed, and revoked. Moreover, function application provides us with a simple and powerful mechanism to model, for instance, cryptographic hash-functions and key-tables.

To illustrate how this mechanism works, let f and k range over constants (in the typed model of type $\mathsf{function}$). As we will see shortly, under the Dolev-Yao model of the intruder that we define, when the intruder knows the constant f, then he can build the hash-value $f(m)$ for any message m he knows. However, just knowing $f(m)$ is not enough to recover m. A similar remark applies for a key-table k of public keys, where every agent a uses $k(a)$ as a public key (so knowing k means knowing the public key of every known agent) and $k(a)^{-1}$ as a private key; this private key is a message initially known by the corresponding agent, but no agent can construct this term.

Observe that there is no syntactic restriction for the message terms that can be used as the first argument of the $\cdot(\cdot)$ operator. Thus, function terms are not treated differently from other message terms and can, for instance, be transmitted as parts of messages.

The IF contains both positive and negative *facts*. A *(positive) fact* represents either the local state of an honest agent, a message in transit through the network (i.e. one sent but not yet received), a message known by the intruder, or a secret message, where $\mathsf{secret}(\mathsf{m},\mathsf{a})$ means that m is a secret and that agent a is allowed to know it. *Negative facts* allow for the modeling of a wider range of protocols than with languages based on standard rewrite rules that manipulate only positive facts. For instance, negative facts allow us to express goals that explicitly require negation, e.g. to state that the intruder does not find out some secret. As a concrete example, to formalize the violation of the secrecy of a message, we could specify the attack-rule $\mathsf{secret}(\mathsf{M},\mathsf{A}).\mathsf{i_knows}(\mathsf{M}).\mathsf{not}(\mathsf{secret}(\mathsf{M},\mathsf{i}))$, which expresses that the intruder i knows some message M that is a secret that some agent A is allowed to know but not the intruder. (Attack-rules are formally defined below.)

A *state* is a finite set of positive (ground) facts, which we denote as a sequence of positive facts separated by dots. Note that in our approach we employ set rewriting instead of multiset rewriting, which is adopted for instance in [19, 20, 26]. Note also that the sets of positive facts and composed messages (i.e. the context-free languages $\mathcal{L}(\mathit{PosFact})$ and $\mathcal{L}(\mathit{ComposedMsg})$) can be easily extended, without affecting the theoretical results that we present below.

To illustrate the benefits of adding negative facts (as well as other fact symbols such as set membership), consider the Needham-Schroeder public key protocol with a key-server [21]. In a realistic model of this protocol, an agent should (i) maintain a database of known public keys, which is shared over all protocol executions that he participates in, and (ii) ask the key-server for the public key of another agent only if this key is not contained in his database. This situation can be directly modeled using negation and an additional fact symbol $\mathsf{knows_pk}$.

Before we explain the remaining parts of the grammar, let us define some standard notions (see, e.g., [6]) and their extensions.

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* $v \in V$. *As we only consider*

[2] 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.

substitutions with finite domains, we 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 we 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*

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

Since the composition of compatible ground substitutions is associative and commutative, so is the $\sqcap$ *operator.*

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).*

Finally, for ϕ *a propositional combination of equalities and for* σ *a substitution for the free variables of* ϕ, *we define the relation* $\sigma \models \phi$ *to represent that* ϕ *is satisfied by* σ *in the structure given by the freely generated term algebra (in our case with the carrier set* $\mathcal{L}(Msg)$*).*

A *condition* is a conjunction of inequalities of messages. Rules describe state transitions. The left-hand side (LHS) of a *Rule* consists of a set of positive facts P, a set of negative facts N, and a condition *Cond*, where $\mathrm{vars}(P) \supseteq \mathrm{vars}(N) \cup \mathrm{vars}(Cond)$. As we will formally define below (Definition 4), a rule is applicable to a state if (i) the positive facts are contained in the state for some substitution σ of the rule's variables, (ii) the negative facts under σ are not contained, and (iii) the condition *Cond* is satisfied under σ. The right-hand side (RHS) of a rule LHS $\Rightarrow$ RHS is just a set of positive facts, where we require that vars(LHS) $\supseteq$ vars(RHS). We will define the successors of a state S as the states generated by replacing in S the facts that match the positive facts of the LHS of some applicable rule with the RHS of that rule.

In this paper, we consider only IF rules of the form

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

where N_1 is a set of negative facts that do not contain i_knows or msg facts, P_1 and P_2 are sets of positive facts that do not contain state or msg facts, and *Cond* is a condition, i.e. a conjunction of inequalities of messages. Moreover, we require that if $\mathsf{i_knows}(m) \in P_1$ then $\mathsf{i_knows}(m) \in P_2$; this ensures that the intruder knowledge is *monotonic*, i.e. that the intruder never forgets messages during transitions.

More specifically, every rule describes a transition of an honest agent, since a state fact appears in both the LHS and the RHS of the rule. Also, in both sides we have a msg fact representing the incoming message that the agent expects to receive in order to make the transition (in the LHS) and the agent's answer message (in the RHS). The rule corresponding to the initial (respectively, final) protocol step contains no incoming (respectively, outgoing) message. However, the rule form (1) is not a restriction here, as one may always insert a dummy message that can be generated by the intruder. In fact, rules of the form (1) are adequate to describe a large class of protocols, including all those discussed in §7.

An *attack-rule* of a protocol description describes the condition under which an attack takes place. We formalize an attack-rule syntactically and semantically like the LHS of a rule of the form (1), with the same restriction on the variables described above. That is, an attack-rule characterizes those states for which a rule with the same LHS is applicable, which we henceforth call *attack-states*. Note that we can always introduce dummy message and state facts so that an attack-rule has the required form (but will refrain from considering dummies in our examples for simplicity).

We now conclude our discussion of the syntax of the IF. A *protocol description ProtocolDescr* is a triple (I, R, AR), consisting of an *initial state* I, a *set* R *of rules*, and a *set* AR *of attack-rules*. A protocol description constitutes a *protocol* when the initial state is ground.

Example 1. As an example, when given the description of the Yahalom protocol of Fig. 1, the HLPSL2IF translator produces an IF file with the following initial state (according to the session instances and the initial knowledge associated to each role):

$$\begin{array}{l}\mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess1},\mathsf{a},\mathsf{b},\mathsf{s},\mathsf{k}(\mathsf{a},\mathsf{s})).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess1},\mathsf{a},\mathsf{b},\mathsf{s},\mathsf{k}(\mathsf{b},\mathsf{s})).\\ \mathsf{state}(\mathsf{roleS},\mathsf{step0},\mathsf{sess1},\mathsf{a},\mathsf{b},\mathsf{s},\mathsf{k}).\\ \mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess2},\mathsf{i},\mathsf{b},\mathsf{s},\mathsf{k}(\mathsf{b},\mathsf{s})).\mathsf{state}(\mathsf{roleS},\mathsf{step0},\mathsf{sess2},\mathsf{i},\mathsf{b},\mathsf{s},\mathsf{k}).\\ \mathsf{i_knows}(\mathsf{a}).\mathsf{i_knows}(\mathsf{b}).\mathsf{i_knows}(\mathsf{s}).\mathsf{i_knows}(\mathsf{i}).\mathsf{i_knows}(\mathsf{k}(\mathsf{i},\mathsf{s}))\quad .\end{array}$$

Note that in the state facts we write, for example, roleA to denote the role A of the protocol, and that, here and in the following, we omit the pairing operator to simplify the notation when no confusion arises. The first three state facts represent the first declared session between the agents a, b, and s, followed by two state facts that represent the second declared session between the intruder i, and the honest agents b and s. Note too that there are only state facts for the honest agents b and s in this session, as the intruder model we give below subsumes the correct execution of the protocol steps by the intruder. The fact $\mathsf{i_knows}(\mathsf{k}(\mathsf{i},\mathsf{s}))$ represents that the intruder has a shared key with the server, which he needs to play in the second session of the protocol. More generally, when a session instance declares the intruder to play a certain role, then all the initial knowledge declared for that role is, under the instantiation, added to the initial intruder knowledge. The second argument of the state facts here indicates the current step number in the protocol execution (which is initially step0), the third argument is a session identifier which is inserted by the HLPSL2IF translator to simplify the generation of fresh values.

To illustrate the transition rules of the honest agents, let us consider only those rules that describe the behavior of an agent in role roleB. In the agent's first transition, he receives the initial message from some agent A containing a nonce NA, he generates a fresh value for the nonce NB, and he sends the appropriate message to the server:

$$\begin{array}{l}\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{SID},\mathsf{A},\mathsf{B},\mathsf{S},\mathsf{KBS}).\\ \mathsf{msg}(\mathsf{A},\mathsf{NA})\\ \Rightarrow\\ \mathsf{state}(\mathsf{roleB},\mathsf{step1},\mathsf{SID},\mathsf{A},\mathsf{B},\mathsf{S},\mathsf{KBS},\mathsf{NA},\mathsf{fresh}(\mathsf{idNB},\mathsf{SID})).\\ \mathsf{msg}(\mathsf{B},\{|\mathsf{A},\mathsf{NA},\mathsf{fresh}(\mathsf{idNB},\mathsf{SID})|\}_{\mathsf{KBS}})\quad .\end{array}$$

In his second transition, the agent playing in roleB receives the third message of the protocol from the agent A and checks that the key contained in the first encrypted part, which seemingly comes from the server, is used to encrypt the nonce NB generated (and stored) by B earlier:

$$\begin{array}{l}\mathsf{state}(\mathsf{roleB},\mathsf{step1},\mathsf{SID},\mathsf{A},\mathsf{B},\mathsf{S},\mathsf{KBS},\mathsf{NA},\mathsf{NB}).\\ \mathsf{msg}(\{|\mathsf{A},\mathsf{KAB}|\}_{\mathsf{KBS}},\{|\mathsf{NB}|\}_{\mathsf{KAB}})\\ \underline{.\mathsf{not}(\mathsf{seen}(\mathsf{B},\mathsf{KAB}))}\\ \Rightarrow\\ \mathsf{state}(\mathsf{roleB},\mathsf{step4},\mathsf{SID},\mathsf{A},\mathsf{B},\mathsf{S},\mathsf{KBS},\mathsf{NA},\mathsf{NB},\mathsf{KAB})\\ \underline{.\mathsf{seen}(\mathsf{B},\mathsf{KAB})}\quad .\end{array}\qquad(2)$$

To make the example also cover negation, we have underlined a possible extension of the rule, which expresses that the honest agent playing in roleB additionally performs a replay check: we introduce a binary fact symbol seen and we express with the underlined fact in the RHS that an agent stores all keys he has seen so far (in any session), while with the underlined fact in the LHS we ensure that he never accepts a key that he has already seen.[3]

[3]One might argue that the nonce NB freshly created by the agent playing in roleB already ensures (without such

Finally, the attack-rule for the specified goal of the Yahalom protocol characterizes the set of states in which the agent playing in the role roleB has finished the protocol, accepting a certain key KAB as generated from the server S for communication between A and B, while the server never issued that key for that purpose:

$$\mathsf{state}(\mathsf{roleB}, \mathsf{step4}, \mathsf{SID}, \mathsf{A}, \mathsf{B}, \mathsf{S}, \mathsf{KBS}, \mathsf{NA}, \mathsf{NB}, \mathsf{KAB}).$$
$$\mathsf{not}(\mathsf{state}(\mathsf{roleS}, \mathsf{step3}, \mathsf{SID}', \mathsf{A}, \mathsf{B}, \mathsf{S}, \mathsf{K}, \mathsf{KAB}))$$

This is exactly the attack-rule that fires in the state reached by the attack-trace given in Fig. 1: an honest agent accepts the pair NA, fresh(idNB,sess2) as the key from the server for communication with the intruder, although the server never issued this key. Note that this attack-rule is automatically generated by the HLPSL2IF translator for the goal `B weakly_authenticates S on KAB`, while the strong authentication goal of Fig. 1 generates an attack-rule that additionally considers replays. A detailed discussion of various kinds of authentication goals can be found in [37]. □

2.3 The Dolev-Yao Intruder

We follow Dolev and Yao [27] and consider the standard model of an active intruder who 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*:*

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

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{m_2\}_{m_1} \in \mathcal{DY}(M)}\ G_{\text{crypt}}\,,$$

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{\!|m_2|\!\}_{m_1} \in \mathcal{DY}(M)}\ G_{\text{scrypt}}\,,$$

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{m_1(m_2) \in \mathcal{DY}(M)}\ G_{\text{apply}}\,,$$

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

$$\frac{\{m_2\}_{m_1} \in \mathcal{DY}(M) \quad {m_1}^{-1} \in \mathcal{DY}(M)}{m_2 \in \mathcal{DY}(M)}\ A_{\text{crypt}}\,,$$

$$\frac{\{m_2\}_{{m_1}^{-1}} \in \mathcal{DY}(M) \quad m_1 \in \mathcal{DY}(M)}{m_2 \in \mathcal{DY}(M)}\ A^{-1}_{\text{crypt}}\,.$$

The generation rules express that the intruder can compose messages from known messages using pairing, asymmetric and symmetric encryption, and function application. The analysis rules describe how the intruder can decompose messages. Note that no rules are given that allow the intruder to analyze function applications, for example to recover m from $f(m)$.

a replay-check) the freshness of the session key KAB, as in the final message NB must be encrypted with KAB. However the replay attack first mentioned in [46] shows that this argumentation is not valid since the message from the agent playing in roleS for the agent playing in roleB in which KAB is issued does not contain NB. Note that the attack of [46] is prevented by this replay-check, while the attack given in Fig. 1 still works.

Moreover, note that this formalization correctly handles non-atomic keys, for instance $m \in \mathcal{DY}(\{\, \{|m|\}_{f(k_1,k_2)}, k_1, k_2, f \,\})$. This is in contrast to other models such as [1, 38, 44, 49] that only handle atomic keys.

2.4 The Semantics of the Intermediate Format

Using $\mathcal{DY}$, we now define a protocol model for the IF in terms of an infinite-state transition system. In this definition, we incorporate an optimization that we call *step-compression*, which is based on the idea [1, 11, 20, 25, 41] 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. More specifically, we compose (or "compress") several steps: when the intruder sends a message, an agent reacts to it according to the agent's rules, and the intruder intercepts the agent's answer.

Definition 4. *Let $r = lhs \Rightarrow rhs$ be a rule of the form* (1)*, i.e.*

$$\mathsf{msg}(m_1).\mathsf{state}(m_2).P_1.N_1 \wedge Cond \Rightarrow \mathsf{state}(m_3).\mathsf{msg}(m_4).P_2\,,$$

and let $\overline{P_1}$ be obtained from P_1 by removing all i_knows *facts, i.e.*

$$\overline{P_1} = P_1 \setminus \{f \mid \exists m.\, f = \mathsf{i_knows}(m)\}\,. \tag{3}$$

We define the applicability of such a rule r by the function applicable that maps a state S and the left-hand side lhs of r to the set of ground substitutions under which the rule can be applied to the state:

$$\begin{aligned}
&applicable_{lhs}(S) = \{\, \sigma \mid \\
&\quad \mathrm{ground}(\sigma) \ \wedge\ \mathrm{dom}(\sigma) = \mathrm{vars}(m_1) \cup \mathrm{vars}(m_2) \cup \mathrm{vars}(P_1) \ \wedge && (4)\\
&\quad \{m_1\sigma\} \cup \{m\sigma \mid \mathsf{i_knows}(m) \in P_1\} \subseteq \mathcal{DY}(\{\, m \mid \mathsf{i_knows}(m) \in S\}) \ \wedge && (5)\\
&\quad \mathsf{state}(m_2\sigma) \in S \ \wedge\ \overline{P_1}\sigma \subseteq S \ \wedge && (6)\\
&\quad (\forall f.\ \mathsf{not}(f) \in N_1 \implies f\sigma \notin S) \ \wedge \sigma \models Cond\ \}\,. && (7)
\end{aligned}$$

We can then define the successor function

$$succ_R(S) = \bigcup_{r \in R} step_r(S)$$

that given a set R of rules of the above form and a state S yields the corresponding set of successor states by means of the following step function:

$$\begin{aligned}
&step_{lhs \Rightarrow rhs}(S) = \{S' \mid \exists\ \sigma.\\
&\quad \sigma \in applicable_{lhs}(S) \ \wedge && (8)\\
&\quad S' = (S \setminus (\mathsf{state}(m_2\sigma) \cup \overline{P_1}\sigma)) \cup \mathsf{state}(m_3\sigma) \cup \mathsf{i_knows}(m_4\sigma) \cup P_2\sigma\,\}\,. && (9)
\end{aligned}$$

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

The function *applicable* yields the set of ground substitutions under which a rule can be applied to a state. In particular, the condition (5) ensures that the message m_1 (that is expected by the honest agent) as well as all messages that appear in i_knows facts in P_1 can be generated from the intruder knowledge under σ, where according to (4) σ is a ground substitution for the variables in the positive facts of the LHS of the rule r. Note that this ensures that each i_knows fact in the LHS of a rule is treated like a message that the intruder has to generate. In particular, the message to generate is not required to be directly contained in the intruder knowledge, but rather it is sufficient that the intruder can generate this message from his knowledge. With $\overline{P_1}$ as defined

by (3) we refer to all facts in P_1 other than $\mathsf{i_knows}$ facts. The conjuncts (6) ensure that the other positive facts of the rule appear in the current state under σ, and (7) ensures that none of the negated facts are contained in the current state under σ, and that the conditions are satisfied under σ.

The step function implements the step-compression technique described above in that it combines three transitions: 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 intercepts this reply and adds it to his knowledge. In particular, the step function creates the set of successor states of a state S by identifying the substitutions such that the given rule is applicable (as is done in condition (8)), and by defining, under such substitutions σ, the successor states S' that result by removing from S the positive facts of the LHS of r and replacing them with the RHS of r (as is done in condition (9)).

Example 2. As an example, we consider the step performed according to the second (extended) rule of the Yahalom protocol for roleB, i.e. (2). We have the following instantiation for the meta-variables in the description of the step-function:

- $m_1 = \{|\mathsf{A}, \mathsf{KAB}|\}_{\mathsf{KBS}}, \{|\mathsf{NB}|\}_{\mathsf{KAB}}$ for the incoming message,
- $m_2 = \mathsf{roleB}, \ldots, \mathsf{NB}$ for the message describing the current local state of the agent playing in roleB,
- $m_3 = \mathsf{roleB}, \ldots, \mathsf{NB}, \mathsf{KAB}$ for the message describing the agent's next state,
- $m_4 = \mathsf{finished}$ for the reply message, where $\mathsf{finished}$ is a dummy message (initially known by the intruder) to give the rule the required form,
- $P_1 = \emptyset$,
- $N_1 = \{\mathsf{not}(\mathsf{seen}(\mathsf{B}, \mathsf{KAB}))\}$,
- $P_2 = \{\mathsf{seen}(\mathsf{B}, \mathsf{KAB})\}$, and
- $Cond = \mathsf{true}$.

Now consider a state S that contains the fact

$$\mathsf{state}(\mathsf{roleB}, \mathsf{step1}, \mathsf{sess2}, \mathsf{i}, \mathsf{b}, \mathsf{s}, \mathsf{k}(\mathsf{b}, \mathsf{s}), \mathsf{na}, \mathsf{fresh}(\mathsf{idNB},\mathsf{sess2}))\ ,$$

where na is a value that the intruder chose earlier. Further, assume that in S the intruder has received from the server the message

$$\{|\mathsf{b}, \mathsf{fresh}(\mathsf{idKAB},\mathsf{sess2}), \mathsf{na}, \mathsf{fresh}(\mathsf{idNB},\mathsf{sess2})|\}_{\mathsf{k(i,s)}},$$
$$\{|\mathsf{i}, \mathsf{fresh}(\mathsf{idKAB},\mathsf{sess2})|\}_{\mathsf{k(b,s)}}\ .$$

Let us refer to the fresh values $\mathsf{fresh}(\mathsf{idNB},\mathsf{sess2})$ and $\mathsf{fresh}(\mathsf{idKAB},\mathsf{sess2})$ as nb and kab for short. Then the successor states of $step_r(S)$ are determined as follows. Let $\sigma = [\mathsf{SID} \mapsto \mathsf{sess2}, \mathsf{A} \mapsto \mathsf{i}, \mathsf{B} \mapsto \mathsf{b}, \mathsf{S} \mapsto \mathsf{s}, \mathsf{KBS} \mapsto \mathsf{k}(\mathsf{b}, \mathsf{s}), \mathsf{NA} \mapsto \mathsf{na}, \mathsf{KAB} \mapsto \mathsf{kab}, \mathsf{NB} \mapsto \mathsf{nb}]$. Two conditions must be satisfied for $step_r(S)$ to yield a successor state with this substitution σ. First, the intruder must be able to generate $m_1\sigma$, which is

$$\{|\mathsf{i}, \mathsf{kab}|\}_{\mathsf{k(b,s)}}, \{|\mathsf{nb}|\}_{\mathsf{kab}}\ .$$

That is, it must be that $m_1\sigma \in \mathcal{DY}(\{m \mid \mathsf{i_knows}(m) \in S\})$. Second, the negative facts under σ must not be contained in S, i.e. it must be that $\mathsf{seen}(\mathsf{b}, \mathsf{kab}) \notin S$. Under these two conditions, the rule r is applicable under σ since, by assumption,

$$\mathsf{state}(m_2\sigma) = \mathsf{state}(\mathsf{roleB}, \mathsf{step1}, \mathsf{sess2}, \mathsf{i}, \mathsf{b}, \mathsf{s}, \mathsf{k}(\mathsf{b}, \mathsf{s}), \mathsf{na}, \mathsf{nb}) \in S\ ,$$

$\overline{P_1}\sigma = P_1\sigma = \emptyset \subseteq S$, and $Cond = \mathsf{true}$. S' is obtained by replacing the matched state fact with the updated fact

$$\mathsf{state}(m_3\sigma) = \mathsf{state}(\mathsf{roleB}, \mathsf{step1}, \mathsf{sess2}, \mathsf{i}, \mathsf{b}, \mathsf{s}, \mathsf{k}(\mathsf{b}, \mathsf{s}), \mathsf{na}, \mathsf{nb}, \mathsf{kab}) \ ,$$

as well as $P_2\sigma = \mathsf{seen}(\mathsf{b}, \mathsf{kab})$. Since the intruder already knows the dummy message $m_4\sigma = \mathsf{finished}$, the intruder knowledge does not grow. □

Definition 5. *We define the* set of reachable states *of a protocol* (I, R, AR) *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 definition of a protocol description and the form of the rules). 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.

We now introduce a predicate $isAttack_{ar}(S)$ that characterizes insecure states: if the attack-rule ar is applicable at state S, then S is an insecure state.

Definition 6. *We define the* attack-predicate $isAttack_{ar}(S)$ *to be true iff* $applicable_{ar}(S) \neq \emptyset$. *We then say that a protocol* (I, R, AR) *is* secure *iff* $isAttack_{ar}(S)$ *is false for all* $S \in reach(I, R)$ *and all attack-rules* $ar \in AR$.

3 The Lazy Infinite-State Approach

In the previous section, we have defined a protocol model for the IF in terms of an infinite-state transition system. This 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 since, by the definition of $\mathcal{DY}$, every node has infinitely many children. It is also of infinite depth, provided we do not bound (and in fact we cannot recursively bound) the number of protocol sessions. The lazy intruder technique presented in the next section uses a symbolic representation to solve the problem of infinite branching, while the *lazy infinite-state approach* [7, 8] allows us to work with infinitely long branches. As we have integrated the lazy intruder with this approach, we now briefly summarize the main ideas of [7, 8].[4]

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 an attack-state. Our on-the-fly model-checker OFMC uses iterative deepening to search this infinite tree. When an attack is found, OFMC returns the attack-trace, i.e. the sequence of exchanged messages leading to the attack-state (cf. Fig. 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 (e.g. using the approach to bounded session generation described in §6) when we employ the lazy intruder to handle the infinite set of messages the intruder can generate.

The lazy approach has several strengths. 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 transition system 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 is then searched. Although semantics, heuristics, and search are all 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 to implement OFMC.

[4]Note that there is no relation between the lazy intruder and the lazy protocol analysis, except that both are demand-driven ("lazy") techniques.

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. The representation is evaluated in a demand-driven way and hence the intruder is called *lazy*.

The idea behind the lazy intruder was, to our knowledge, first proposed by [32] and then subsequently developed by [1, 11, 12, 19, 25, 31, 41]; see [22] for an overview. Our contributions to the lazy intruder technique are as follows. First, we simplify the technique, which, as we show in the appendix, also leads to a simpler proof of its correctness and completeness. Second, we formalize its integration into the search procedure induced by the rewriting approach of the IF and, on the practical side, we present (in §5) an efficient way to organize and implement the combination of state exploration and constraint reduction. Third, we extend the technique to ease the specification and analysis of a larger class of protocols and properties, where we implement negative facts and conditions in the IF rewrite rules by inequality constraints for the lazy intruder. Finally, we show how to employ the lazy intruder to solve the problem of instantiating protocols for particular analysis scenarios (cf. §6).

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. Therefore, 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 that part by replacing it with a variable and recording a constraint on which knowledge the intruder can use 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").

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.*

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

We say that 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 a finite set of constraints and its semantics is the intersection of the semantics of its elements, i.e., overloading notation, $[\![\{c_1, \ldots, c_n\}]\!] = \sqcap_{i=1}^{n} [\![c_i]\!]$. A constraint set C is* satisfiable *if $[\![C]\!] \neq \emptyset$. A constraint set C is* simple *if all its constraints are simple, and we then write* $\mathrm{simple}(C)$.

Example 3. As an example, consider again the trace of the attack on the Yahalom protocol in Fig. 1, and let us again refer to the fresh values $\mathsf{fresh}(\mathsf{idNB},\mathsf{sess2})$ and $\mathsf{fresh}(\mathsf{idKAB},\mathsf{sess2})$ as nb and kab for short. The intruder first chooses a nonce NA for communication with b. Then, the intruder sees both the message from b to s, namely $\{\!|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\!\}_{\mathsf{k(b,s)}}$, and the message from s to i, namely $\{\!|\mathsf{b}, \mathsf{kab}, \mathsf{NA}, \mathsf{nb}|\!\}_{\mathsf{k(i,s)}}, \{\!|\mathsf{i}, \mathsf{kab}|\!\}_{\mathsf{k(b,s)}}$. Hence, the following constraints arise from the steps taken in the trace:

$$\begin{array}{l}\{\,\mathit{from}(\mathsf{NA}, IK_0)\,,\\ \quad \mathit{from}(\langle \{\!|\mathsf{i}, \mathsf{KAB}|\!\}_{\mathsf{k(b,s)}}, \{\!|\mathsf{nb}|\!\}_{\mathsf{KAB}}\rangle\,,\\ \quad\quad IK_0 \cup \{\!|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\!\}_{\mathsf{k(b,s)}} \cup \{\!|\mathsf{b}, \mathsf{kab}, \mathsf{NA}, \mathsf{nb}|\!\}_{\mathsf{k(i,s)}}, \{\!|\mathsf{i}, \mathsf{kab}|\!\}_{\mathsf{k(b,s)}})\,\}\,,\end{array}$$

where KAB is a fresh variable and IK_0 is the initial intruder knowledge, which includes all agent names and the intruder's shared key with the server, $\mathsf{k(i,s)}$. □

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

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

$$\frac{\mathit{from}(m_1 \cup m_2 \cup T, IK) \cup C, \sigma}{\mathit{from}(\{m_2\}_{m_1} \cup T, IK) \cup C, \sigma}\ G^l_{\text{crypt}}\,,$$

$$\frac{\mathit{from}(m_1 \cup m_2 \cup T, IK) \cup C, \sigma}{\mathit{from}(m_1(m_2) \cup T, IK) \cup C, \sigma}\ G^l_{\text{apply}}\,,$$

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

$$\frac{\mathit{from}(T, m_1 \cup m_2 \cup \langle m_1,m_2\rangle \cup IK) \cup C, \sigma}{\mathit{from}(T, \langle m_1,m_2\rangle \cup IK) \cup C, \sigma}\ A^l_{\text{pair}}\ (\{m_1, m_2\} \setminus IK \neq \emptyset)\,,$$

$$\frac{\mathit{from}(m_1, IK) \cup \mathit{from}(T, m_2 \cup \{\!|m_2|\!\}_{m_1} \cup IK) \cup C, \sigma}{\mathit{from}(T, \{\!|m_2|\!\}_{m_1} \cup IK) \cup C, \sigma}\ A^l_{\text{scrypt}}\ (m_2 \notin IK)\,,$$

$$\frac{\mathit{from}({m_1}^{-1}, IK) \cup \mathit{from}(T, m_2 \cup \{m_2\}_{m_1} \cup IK) \cup C, \sigma}{\mathit{from}(T, \{m_2\}_{m_1} \cup IK) \cup C, \sigma}\ A^l_{\text{crypt}}\ (m_2 \notin IK)\,,$$

$$\frac{\mathit{from}(m_1, IK) \cup \mathit{from}(T, m_2 \cup \{m_2\}_{m_1} \cup IK) \cup C, \sigma}{\mathit{from}(T, \{m_2\}_{{m_1}^{-1}} \cup IK) \cup C, \sigma}\ A^l_{\text{crypt}^{-1}}\ (m_2 \notin IK)\,.$$

Figure 2: Lazy intruder: constraint reduction rules.

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 is performed using the generation and analysis rules of Fig. 2, which describe possible transformations of the constraint set. 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.

A generation or analysis rule r has the form

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

with C and C' constraint sets and σ and σ' substitutions. It expresses that (C', σ') can be *derived* from (C, σ), which we denote by $(C, \sigma) \vdash_r (C', \sigma')$. That is, the constraint reduction rules are applied backwards. Note that σ' extends σ in all rules. As a consequence, we will be able to apply the substitutions generated during the reduction of C also to the facts of a lazy state, as we discuss below.

The generation rules G^l_{pair}, G^l_{scrypt}, G^l_{crypt}, and G^l_{apply} express that the constraint stating that the intruder can generate a message composed from submessages m_1 and m_2 (using pairing, symmetric and asymmetric encryption, and function application, respectively) 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 provided this message can be unified with the message m_1 that 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: how the intruder chooses to instantiate this variable is immaterial at this point in 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 since messages may now contain variables. In particular, if the key term of an encrypted message contains 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 in the key term can be instantiated during subsequent constraint reduction.[5] More specifically, for a message $\{\!|m_2|\!\}_{m_1}$ that the intruder attempts to decrypt, we add the content m_2 to the intruder knowledge of the respective constraint (as if the check was already successful) and add a new constraint expressing that the symmetric key m_1 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 $\{\!|m_2|\!\}_{m_1}$ to be analyzed may not be used in the generation of the key; this is in contrast to similar approaches that can also handle non-atomic symmetric keys, such as [41, 20]. In our notation, their decryption rule is

$$\frac{\mathit{from}(m_1, \{\!|m_2|\!\}_{m_1}{}^* \cup IK) \cup \mathit{from}(T, m_2 \cup \{\!|m_2|\!\}_{m_1} \cup IK) \cup C, \sigma}{\mathit{from}(T, \{\!|m_2|\!\}_{m_1} \cup IK) \cup C, \sigma} \; A^l_{\text{scrypt}}{}^* .$$

This rule is the same as ours, except that the constraint governing the derivation of the key m_1 additionally contains the message $\{\!|m_2|\!\}_{m_1}$ marked with a $*$. This marking denotes that $\{\!|m_2|\!\}_{m_1}$ may not be further analyzed (as there is already an analysis of this term in progress). Without this mark, the approaches of [41, 20] would not terminate since, in the derivation of m_1, one could infinitely often decrypt $\{\!|m_2|\!\}_{m_1}$, repeatedly producing the same constraint. Although the mark ensures termination, it gives the rule a procedural aspect, making it less declarative.

As formally justified in the proof of our completeness theorem (the proof of Theorem 1 in the appendix), our rule A^l_{scrypt}, which omits marking entirely, does not exclude any solution. The intuition behind this is as follows: the only case in which the marked term $\{\!|m_2|\!\}_{m_1}$ is actually used to derive m_1 is when there is another term $t \in IK$ that is encrypted with the term $\{\!|m_2|\!\}_{m_1}$ as a key. However, in this case, we could have first performed the analysis of t and hence need not perform it during the derivation of m_1. In general, if one performs the analysis steps in the order that they depend on each other, no analysis is needed in the constraints that are introduced by the analysis rules, in this case the $\mathit{from}(m_1, \cdot)$ constraint.

Note that our rule is not only simpler and more declarative, it also considerably simplifies the completeness proof. For example, the respective completeness proof in [41] must split into one part with *encryption hiding* (as they call the marked terms) and one without.

Definition 8. *Let $\vdash$ denote the reflexive and transitive closure of the union of the derivation relations $\vdash_r$ for every rule r of Fig. 2. The set of pairs of simple constraint sets and substitutions derivable from (C, id) is $\mathit{Red}(C) = \{(C', \sigma) \mid ((C, \mathrm{id}) \vdash (C', \sigma)) \wedge \mathrm{simple}(C')\}$, where we define $[\![\mathit{Red}(C)]\!] = \{\sigma\sigma' \mid \exists C'.\ (C', \sigma) \in \mathit{Red}(C) \wedge \sigma' \in [\![C']\!]\}$.*

Example 4. Consider the reductions performed on the constraints of the Yahalom example above. First, the intruder can perform an analysis step on the intruder knowledge IK, since a part of the message sent by the server s is encrypted by the shared key $\mathsf{k(i,s)}$ of i and s. Applying the rules A^l_{pair} and A^l_{scrypt} to the second constraint results in the following constraint set:

$$\begin{aligned}\{\quad & \mathit{from}(\mathsf{NA}, IK_0)\,, \\ & \mathit{from}(\mathsf{k(i,s)}, IK_0 \cup \{\!|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\!\}_{\mathsf{k(b,s)}} \cup \{\!|\mathsf{i}, \mathsf{kab}|\!\}_{\mathsf{k(b,s)}})\,, \\ & \mathit{from}(\langle \{\!|\mathsf{i}, \mathsf{KAB}|\!\}_{\mathsf{k(b,s)}}, \{\!|\mathsf{nb}|\!\}_{\mathsf{KAB}} \rangle, \\ & \qquad IK_0 \cup \{\!|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\!\}_{\mathsf{k(b,s)}} \cup \{\!|\mathsf{i}, \mathsf{kab}|\!\}_{\mathsf{k(b,s)}} \cup \mathsf{kab} \cup \mathsf{NA} \cup \mathsf{nb})\ \}\,.\end{aligned}$$

[5] 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.

In the following, we will refer to the intruder knowledge of the third constraint in the above set as IK. The second of these constraints is directly solvable using the G^l_{unif} rule since $\mathsf{k}(\mathsf{i},\mathsf{s}) \in IK_0$. Applying G^l_{pair} to the third constraint replaces the pair in the terms to generate with its components:

$$\{ \quad \mathit{from}(\mathsf{NA}, IK_0)\,, \\ \quad \mathit{from}(\{\!|\mathsf{i}, \mathsf{KAB}|\!\}_{\mathsf{k(b,s)}} \cup \{\!|\mathsf{nb}|\!\}_{\mathsf{KAB}}, IK)\ \}\,,$$

where, here and in the following, we omit the constraints of the form $\mathit{from}(\emptyset, IK)$.

For the first message that the intruder has to generate in the second constraint, i.e. $\{\!|\mathsf{i}, \mathsf{KAB}|\!\}_{\mathsf{k(b,s)}}$, there are two possibilities: using the G^l_{unif} rule, this message can be unified either with the message $\{\!|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\!\}_{\mathsf{k(b,s)}}$ sent earlier by b (where the unifier is $\mathsf{KAB} \mapsto \langle \mathsf{NA},\mathsf{nb}\rangle$), or with the original message $\{\!|\mathsf{i}, \mathsf{kab}|\!\}_{\mathsf{k(b,s)}}$ from the server (where the unifier is $\mathsf{KAB} \mapsto \mathsf{kab}$). The second possibility reflects the "correct" protocol execution (and the remaining constraint is easily solved in this case). Let us thus consider the other possibility, which leads to the attack displayed in Fig. 1, i.e. $\mathsf{KAB} \mapsto \langle \mathsf{NA},\mathsf{nb}\rangle$ so that:

$$\{ \quad \mathit{from}(\mathsf{NA}, IK_0)\,, \\ \quad \mathit{from}(\{\!|\mathsf{nb}|\!\}_{\langle \mathsf{NA},\mathsf{nb}\rangle}, IK)\ \}\,.$$

These constraints can be solved by first applying the rules G^l_{crypt} and G^l_{pair}, resulting in

$$\{ \quad \mathit{from}(\mathsf{NA}, IK_0)\,, \\ \quad \mathit{from}(\mathsf{NA} \cup \mathsf{nb}, IK)\ \}\,,$$

and then eliminating nb using the G^l_{unif} rule (as $\mathsf{nb} \in IK$). The remaining constraint set is simple.

To summarize, there are two simple constraint sets corresponding to the original constraint set in this example: one corresponding to the correct execution of the protocol, the other representing an attack. □

4.3 Properties of *Red*

By Theorem 1 below, the *Red* function is correct, complete, and recursively computable (since $\vdash$ is finitely branching). 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. 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:

$$IK_i \quad \subseteq \quad IK_j \ \ \textit{for } i \le j\,, \tag{10}$$

$$\mathrm{vars}(IK_i) \quad \subseteq \quad \bigcup_{j=1}^{i-1} \mathrm{vars}(T_j)\,. \tag{11}$$

Intuitively, (10) requires that the intruder knowledge increases monotonically, and (11) requires that every variable that appears in terms known by the intruder is part of a message that the intruder created earlier. Said another way, variables only "originate" from the intruder.

Note that the analysis rules of the lazy intruder can destroy property (10), as a 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 in the proof of Theorem 1 given in the appendix, there is a straightforward procedure that transforms every simple constraint set obtained by *Red* into an equivalent, well-formed, simple one.

Theorem 1. *Let C be a well-formed constraint set. $\mathit{Red}(C)$ is finite and $\vdash$ is well-founded. Moreover, $[\![C]\!] = [\![\mathit{Red}(C)]\!]$, i.e. $\mathit{Red}(C)$ is correct and complete.*

The intuition behind this theorem is that with every reduction step the constraints become simpler in some sense, and thus $\vdash$ is well-founded and $Red(C)$ is finite. Correctness, i.e. $[\![C]\!] \supseteq [\![Red(C)]\!]$, holds as no rule application adds solutions to the constraint set. Completeness, i.e. $[\![C]\!] \subseteq [\![Red(C)]\!]$, holds as if a solution σ is allowed by the constraint set (i.e. $\sigma \in [\![C]\!]$), then we can either find an applicable rule such that the resulting constraint set C' still supports σ (i.e. $\sigma \in [\![C']\!]$) or C' is already simple. Since $\vdash$ is well-founded, after finitely many applications of rules supporting σ, the resulting constraint set C'' must be simple. Thus if $\sigma \in [\![C]\!]$ then there is a simple $C'' \in Red(C)$ such that $\sigma \in [\![C'']\!]$.

4.4 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 variables (as opposed to the ground model where all reachable states are ground) and that are associated with a set of *from* constraints as well as a collection of inequalities. The inequalities are used to handle negative facts and conditions in the context of the lazy intruder. We require that the inequalities are given as a conjunction of disjunctions of inequalities between terms. We will use the inequalities to rule out certain unifications, for example 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 $(v_1 \neq t_1 \mathbin{\mathsf{V}} v_2 \neq t_2) \mathbin{\mathsf{\wedge}} (v_1 \neq t_3)$. Note that we write V and $\mathsf{\wedge}$ 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 $[\![(P, C, N)]\!] = \{P\sigma \mid \sigma \in [\![C]\!] \wedge \sigma \models N\}$.*

Let $\mathrm{freshvars}_S(r)$ *be a rule obtained from the rule r by renaming the variables in r with respect to the lazy state $S = (P, C, N)$ so that* $\mathrm{vars}(S)$ *and* $\mathrm{vars}(\mathrm{freshvars}_S(r))$ *are disjoint. As in the ground case, let $r = lhs \Rightarrow rhs$ be a rule of the form* (1)*, i.e.*

$$\mathsf{msg}(m_1).\mathsf{state}(m_2).P_1.N_1 \mathbin{\mathsf{\wedge}} \mathit{Cond} \Rightarrow \mathsf{state}(m_3).\mathsf{msg}(m_4).P_2\,,$$

and let $\overline{P_1}$ be obtained from P_1 by removing all i_knows *facts, i.e.*

$$\overline{P_1} = P_1 \setminus \{f \mid \exists m.\, f = \mathsf{i_knows}(m)\}\,. \tag{12}$$

We can then define the applicability of such a rule r to a lazy state (P, C, N) by the function $applicable^l$ that maps (P, C, N) and the left-hand side lhs of r to a set of substitutions under which the rule can be applied to the state:

$$\begin{aligned}
&\mathit{applicable}^l_{lhs}(P, C, N) = \Big\{(\sigma, C', N') \mid \\
&\quad \mathrm{dom}(\sigma) \subseteq \mathrm{vars}(m_1) \cup \mathrm{vars}(m_2) \cup \mathrm{vars}(P_1) \cup \mathrm{vars}(P, C, N) \wedge \\
&\quad \mathsf{state}(m_2\sigma) \in P\sigma \;\wedge \overline{P_1}\sigma \subseteq P\sigma \;\wedge && (13)\\
&\quad C' = (C \cup \\
&\qquad \mathit{from}(m_1 \cup \{m \mid \mathsf{i_knows}(m) \in P_1\}, \{i \mid \mathsf{i_knows}(i) \in P\}))\sigma \wedge && (14)\\
&\quad N' = N\sigma \mathbin{\mathsf{\wedge}} \textstyle\bigwedge_{\phi \in subCont(N_1\sigma, P\sigma)} \phi \mathbin{\mathsf{\wedge}} \mathit{Cond}\,\sigma \;\Big\} && (15)
\end{aligned}$$

where

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

We can then define the lazy successor function

$$succ^l_R(S) = \bigcup_{r \in R} step^l_{\mathrm{freshvars}_S(r)}(S)$$

that maps a set R of rules of the form (1) *and a lazy state $S = (P, C, N)$ to a set of lazy states by means of the following* lazy step function*:*

$$\begin{aligned} step^l_r(P, C, N) = \{\, (P', C', N') \mid \exists\, \sigma. \\ (\sigma, C', N') \in applicable^l_{lhs}(P, C, N) \;\wedge \qquad (16) \\ P' = (P\sigma \setminus (\mathsf{state}(m_2\sigma) \cup \overline{P_1}\sigma)) \;\cup\; \mathsf{state}(m_3\sigma) \cup \mathsf{i_knows}(m_4\sigma) \cup \\ P_2\sigma \,\}. \qquad (17) \end{aligned}$$

The function $applicable^l$ is the "lazy equivalent" of the ground *applicable* function: given a lazy state and the LHS of a rule of the form (1), $applicable^l$ yields the set of triples of substitutions, constraint sets, and inequalities such that the following three conditions are satisfied. Condition (13) is similar to the first two conjuncts in condition (7) in the ground model, where the substitution is now applied also to the set of positive facts in the state (instead of matching, we now perform unification). The constraint in condition (14) expresses that both the message m_1 and i_knows facts of the positive facts of the LHS of the rule r must be generated by the intruder from his current knowledge. Condition (15) states that the inequalities are conjoined with the inequalities of the rule and with the conjunction of all formulae that $subCont(N_1\sigma, P\sigma)$ yields. The name *subCont* represents that this function produces a formula that excludes those most general *sub*stitutions under which the given negative facts are *cont*ained in the given state. More concretely, for a set N of negative facts and a set P of positive facts, $subCont(N, P)$ 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, as is standard, we define $\bigvee_{i=1}^{0} \phi$ to be false for any ϕ. Hence, $subCont(\mathsf{not}(f) \cup N, f \cup P) = \mathsf{false}$, for any fact f. Also N' is conjoined with the inequalities of the rule under σ. Note that unlike in the ground model, we cannot directly check here if the condition is satisfied since it is not necessarily a ground term; instead, we store this constraint.

Similar to the successor function of the ground model, the lazy successor function also performs step-compression, by exploiting the lazy step function $step^l$ (where, in contrast to the ground case, we rename the variables of the rule to avoid clashes with the variables that may appear in the lazy state). The lazy step function $step^l$ creates the set of lazy successor states of a lazy state (P, C, N) by first using the $applicable^l$ function to identify triples consisting of the new constraints C', the new inequalities N', and a substitution σ such that the given rule is applicable (as is done in condition (16)), and then using this σ to describe 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.

Note that the lazy applicability and step functions do not check the satisfiability of the generated constraints and inequalities. This is because we do not want to prescribe, as part of the formalism, whether or not constraints are directly reduced after every transition. Instead, we leave this decision to the search strategy, discussed in §5.

Example 5. We return to our Yahalom example to contrast the lazy successor function $step^l$ with the ground successor function *step*. The major difference is that we now start with a symbolic state (P, C, N) where $S \in [\![(P, C, N)]\!]$ for the state S introduced in Example 4. We can obtain such a symbolic state, occurring in the analysis of the Yahalom protocol, by replacing in S the value na (whatever the agent playing in roleB received) with the variable NA (this yields the set P of positive symbolic facts) and the constraint set $C = \{from(\mathsf{NA}, IK_0)\}$ representing that the intruder generated this value NA earlier.

The substitution taken in the $step^l$ function thus differs from the substitution taken in the ground case *step* by not substituting values for NA and for KAB. Note that the value of KAB is

not determined by the rule since the agent playing in roleB accepts any value whenever he receives messages of the proper format. For instance,

$$m_1\sigma = \{|\mathsf{i}, \mathsf{KAB}|\}_{\mathsf{k(b,s)}}, \{|\mathsf{NB}|\}_{\mathsf{KAB}} \ .$$

Another difference is that the $step^l$ function does not "check" that the intruder can generate the messages in question (in this case $m_1\sigma$). Instead, it adds an appropriate constraint to the constraint store (in this case $C' = C \cup from(m_1\sigma, IK)$, where IK is the set of messages m for which $\mathsf{i_knows}(m) \in P$). If there is no solution for this constraint, then the semantics of the successor state (P', C', N') is empty.

Similarly, the negative facts are not directly evaluated. Suppose, for instance, that the set P contains the fact $\mathsf{seen}(\mathsf{b}, \mathsf{kab})$. Then the new conjunctions added to N by $subCont(N_1\sigma, P\sigma)$ (where $N_1\sigma$ is the condition $\mathsf{not}(\mathsf{seen}(\mathsf{b}, \mathsf{KAB}))$) entail the inequality $\mathsf{KAB} \neq \mathsf{kab}$. Intuitively, the newly received key must differ from all keys already in the database of seen keys and this check is done as soon as the constraint set is reduced, possibly leading to substitutions for the variable KAB.

Finally, in the successor state (P', C', N') we have the updated state fact for b, containing now both the variables NA and KAB, and the (non-simple) constraint set C' as described above. □

Definition 11. *We define the set of* reachable lazy states *of a protocol* (I, R, AR) *as* $reach^l(I, R) = \bigcup_{n\in\mathbb{N}} (succ^l_R)^n(I, \emptyset, \emptyset)$.

We also call $reach^l(I, R)$ the *lazy intruder model* of the protocol (I, R, AR), or the *lazy model* for short.

As we show in the appendix, the lazy model is equivalent to the ground model, in the sense that they both represent the same set of reachable states.

Lemma 1. $reach(I, R) = \cup_{(P,C,N)\in reach^l(I,R)} [\![(P, C, N)]\!]$ *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 $isAttack_{ar}(S)$ is false for all reachable ground states S and $ar \in AR$. A similar check suffices in the lazy intruder model, where we rename the variables of the attack-rule as for the lazy successor function in Definition 10 in order to avoid clashes.

Definition 12. *For a lazy state* (P, C, N) *and an attack-rule* ar*, we define the lazy attack-predicate* $isAttack^l_{ar}(P, C, N)$ *to be true iff* $[\![(P\sigma, C', N')]\!] \neq \emptyset$ *for some* $(\sigma, C', N') \in applicable^l_{ar'}(P, C, N)$ *with* $ar' = \text{freshvars}_{(P,C,N)}(ar)$.

If $isAttack^l$ is true for a reachable lazy state (P, C, N), then (P, C, N) represents an attack-state:

Lemma 2. *For all lazy states* (P, C, N) *and all attack-rules* ar*, the predicate* $isAttack^l_{ar}(P, C, N)$ *holds iff* $isAttack_{ar}(S)$ *holds for some represented ground state* $S \in [\![(P, C, N)]\!]$.

We thus have the following theorem, proven in the appendix.

Theorem 2. *A protocol* (I, R, AR) *is* secure *iff* $isAttack^l_{ar}(P, C, N)$ *is false for all attack-rules* $ar \in AR$ *and all reachable lazy states* $(P, C, N) \in reach^l(I, R)$.

Using the above results, we now show how the lazy intruder allows us to build an effective decision procedure for protocol (in)security for a bounded number of sessions with unbounded message complexity, and a semi-decision procedure for protocol insecurity for an unbounded number of sessions. To this end, we have to tackle three problems.

First, the $step^l$ function may return infinitely many successors, as there can be infinitely many unifiers σ for the positive facts of the rules with a 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, represent only 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 provides a straightforward solution to this problem: we represent the reachable states as the tree generated using the lazy intruder successor function. For an unbounded number of sessions, this tree is infinitely deep, but using lazy data-types we can compute with a finite representation of it. In particular, we can apply the lazy attack-predicate as a filter on this tree to obtain the lazy attack-states.

Third, we must check whether one of these lazy attack-states is satisfiable, i.e. represents a possible attack. (We will see that this check can also be applied as a filter on the tree.) Constraint reduction is the key to this task. By Theorem 1, we know that, for a well-formed constraint set C, constraint 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.

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

The proof given in the appendix is based on the observation that a simple constraint set with inequalities is always satisfiable as the intruder can always generate sufficiently many different messages. This is the intuition why inequalities can be so easily integrated into our lazy model.

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 $\mathit{Red}(C) = \emptyset$ 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 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. This follows from the way that new constraints are generated during the step^l transitions.

Lemma 4. *For a protocol (I, R, AR), if $(P, C, N) \in \mathit{reach}^l(I, R)$ then C is well-formed.*

We can now put all the pieces together to obtain an effective procedure for checking whether a protocol is secure: we generate reachable lazy states and filter them both for attack-states and for constraint satisfiability. We next describe how to implement this procedure in an efficient way.

5 Organizing State Exploration and Constraint Reduction

When implementing the lazy intruder, we are faced with two design decisions: (i) in which order to apply the two "filters" mentioned above, and (ii) how to realize constraint reduction.

With respect to (i), 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. OFMC applies *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 will again have unsatisfiable constraint sets. We also extend this idea to checking the inequalities and remove states with unsatisfiable inequalities. In the lazy infinite-state approach, this can be realized simply 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 pruning several subtrees, and then for attack-states (using $\mathit{isAttack}^l$). Note that *Red* can lead to case splits if there are several solutions for the given constraint set; in this case, to avoid additional branching in the search tree, one can continue the search with the original constraint set.

With respect to (ii), note that the question of how to reduce constraints (in particular, how to analyze the intruder knowledge) is often neglected in other presentations of symbolic intruder approaches. One solution is to proceed on demand: a message in the intruder knowledge is analyzed iff the result of this analysis 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 until a fixed-point is reached, i.e. no rule produces additional knowledge. The result is that the intruder knowledge is "normalized" with respect to the analysis rules. As a consequence, we need not further consider analysis rules during the reduction of the constraints. 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 appears that can be unified with m. In contrast, with analysis on demand it is necessary in this case to check if a unifiable message may be obtained through analysis.

However, when normalizing the intruder knowledge, we must take into account that the analysis may produce substitutions. Every substitution restricts the set of possible solutions and in this case the restriction is only necessary if the respective decrypted message content is actually used later.[6] Our solution is that when an analysis step is possible only under a substitution σ, then we perform a case split. In one case we apply σ and perform the analysis step, and in the other case we do not perform the analysis step and exclude the substitution σ to prevent repeated application of the same case split to this case. Excluding a substitution $\sigma = \{(v_1, t_1), \ldots, (v_n, t_n)\}$ is achieved by conjoining the disjunction of the inequalities

$$v_1 \neq t_1 \vee \ldots \vee v_n \neq t_n$$

to the constraint set.

Note that a demand-driven analysis would perform such a case split only if the analyzed term is actually used. However, our strategy is often more efficient as it does not require complicated caching strategies, and experience shows that most case splits during normalization analysis distinguish states that must also be considered by demand-driven analysis.

6 Symbolic Sessions

The lazy approach to model-checking security protocols described above allows one to work even with an unbounded number of sessions. However, in practice it is often advantageous to organize and control search by considering (and searching) different scenarios under which a protocol should be checked, corresponding to different sessions where different agents assume different roles in the interleaved protocol executions. In this section, we consider several alternatives of increasing sophistication and power: (i) manual session specification, which is common in many tools, e.g. in previous versions of HLPSL and OFMC [2] or CAPSL/CIL [26], (ii) automatic session generation, which has been developed in [14], and finally (iii) our new approach based on symbolic sessions.

6.1 Manual Session Specification

The first alternative is that the user explicitly describes the scenario under which the protocol should be checked using OFMC. This may be done by specifying a finite list of instantiations of the roles of the protocol with agent names, where `i` denotes the intruder and all other agents (`a`, `b`, ...) are honest. For example, in a protocol with roles `A` and `B` (in the notation of the HLPSL, which correspond to roleA and roleB in the notation of the IF) one might specify the following instances:

```
[ A:a, B:b ]
[ A:a, B:b ]
[ A:b, B:i ]  .
```

[6] As an example, suppose that the intruder wants to analyze the message $\{|\{|\mathsf{m}|\}_\mathsf{k}|\}_{\{|\mathsf{M}|\}_\mathsf{k}}$, where the variable M represents a message the intruder 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 unnecessarily limit the possible messages the intruder could have said.

This describes a scenario consisting of three parallel sessions, two between the honest agents a and b, and one between b and the intruder. Recall that the intruder can always send messages under any identity; however, a session instance between an honest agent and the intruder explicitly models an honest agent who runs a protocol with a dishonest or corrupted agent.[7]

The formal meaning of such a scenario is defined by the translation to the IF that is performed by the HLPSL2IF translator. In the IF, the initial state determines the scenario to be explored: every state fact represents an honest agent who is willing to perform one run of the protocol. This is because every rule in the IF contains exactly one state fact both on the left-hand side and on the right-hand side, so that the number of agent facts is the same in all reachable states. A state fact of the initial state thus corresponds to the initial node of a strand in a strand space model or an agent process in a process calculus model; see, e.g., [30, 38, 48] as well the discussions in §8.[8]

Given a protocol with r roles in HLPSL, for every specified session, the HLPSL2IF translator generates r state facts in the initial state of the resulting IF files, one fact for each role that contains the agent names of the respective session instance. For example, for the instances specified above, the initial state contains the facts

$$\begin{aligned}&\mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess1},\mathsf{a},\mathsf{b}).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess1},\mathsf{b},\mathsf{a}).\\&\mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess2},\mathsf{a},\mathsf{b}).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess2},\mathsf{b},\mathsf{a}).\\&\mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess3},\mathsf{b},\mathsf{i}).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess3},\mathsf{i},\mathsf{b})\ ,\end{aligned}$$

where we again omitted the pairing operator to simplify the notation. Every pair of state facts contains a unique identifier sess$j$. In our *set* rewriting approach, this allows us to specify multiple parallel sessions between the same agents (in the example, the sessions sess1 and sess2).

The last state fact, $\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess3},\mathsf{i},\mathsf{b})$, represents that the intruder can execute the protocol in the role roleB with b in the role roleA. This and other state facts for the intruder are actually not necessary under the Dolev-Yao intruder model since an intruder can always execute a protocol in the intended way under his own, real name. Indeed, the HLPSL2IF translator checks that the HLPSL specification of the protocol is executable in the sense that, given the required initial knowledge, every agent can construct the messages he is supposed to, and this ability is of course subsumed by the abilities of the Dolev-Yao intruder.

Manual session instantiation constitutes a basic mechanism for specifying protocol analysis scenarios. However, it is unsatisfactory that the user must specify instances manually. In practice, to avoid state-space explosion due to parallelism, it is desirable to iterate through many scenarios, with differing role instances. Support for searching among instances is called for here.

6.2 Automatic Session Generation

The second alternative is to automatically generate scenarios. This may be accomplished, for a given number n of sessions and a set of roles, by generating all scenarios with n session instances of the roles up to isomorphy, i.e. renaming of the honest agents and re-ordering the list of sessions (see [14]). OFMC can then analyze the protocol for each of the scenarios generated this way. However, even for a small number n, the number of scenarios to be considered is enormous.

6.3 Symbolic Session Generation

One can interpret automatic session generation as a kind of parameter search: the protocol model is parametrized over a scenario and we can explore the values of this parameter (for a given upper bound). We now introduce a refinement of this idea that we use in OFMC. Namely, we improve upon this approach by letting the lazy intruder take care of sessions. This is possible as there is no

[7] This also reflects that we do not distinguish between different intruders and corrupted parties, but assume that they all work together and can thus be merged into one intruder.

[8] We assume here that the rules of an IF protocol specification ensure that every state fact can be involved only in a bounded number of transitions. This assumption holds, for instance, for all protocols that can be specified in HLPSL, but it excludes streaming and group protocols (unless one bounds the length of the stream or the size of the group).

essential difference between choosing an agent from a limited set of possibilities when generating sessions and using the lazy intruder to choose a message from his knowledge during the normal search.

More in detail, we take advantage of the symbolic representation of the lazy intruder to avoid enumerating all possible session instances (for a given bound on the number of sessions). To do this, we instantiate the roles with variables rather than with constant agent names. The variables are then instantiated by unification during search, either to constant agent names or to other variables. For instance, for $n = 3$ and a protocol with two roles roleA and roleB, the lazy initial state contains

$$\begin{array}{c}\mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess1},\mathsf{A}_1,\mathsf{B}_1).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess1},\mathsf{B}_2,\mathsf{A}_2).\\ \mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess2},\mathsf{A}_3,\mathsf{B}_3).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess2},\mathsf{B}_4,\mathsf{A}_4).\\ \mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess3},\mathsf{A}_5,\mathsf{B}_5).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess3},\mathsf{B}_6,\mathsf{A}_6)\\ \mathsf{A}_1 \neq \mathsf{i} \wedge \mathsf{B}_2 \neq \mathsf{i} \wedge \mathsf{A}_3 \neq \mathsf{i} \wedge \mathsf{B}_4 \neq \mathsf{i} \wedge \mathsf{A}_5 \neq \mathsf{i} \wedge \mathsf{B}_6 \neq \mathsf{i} \end{array} \quad .$$

Note that, following our previous discussion, we add inequalities that explicitly prevent unifying variables that are intended to represent honest agents with the name of the intruder.[9]

Now let $\mathsf{Agent} = \{\mathsf{i}, \mathsf{a}, \mathsf{b}, \ldots\}$ be a set of agent names. If all the variables that we have introduced into the initial state range over the set Agent, then this symbolic initial state represents the set of all initial states that would result from automatic session generation with this set of agents (for a given bound n of the sessions).

In order to integrate the approach just described with the lazy intruder there is, however, one subtlety that we must address: we have assumed that all constraint sets are well-formed, in particular, that each variable that appears in a constraint set is introduced on the left-hand side of some constraint. Intuitively speaking, combining session instantiation with the lazy intruder means that the intruder chooses the concrete names of the agents in the initial state, but leaves variables for these choices and lazily instantiates them during the search. The initial state must therefore contain a constraint set that requires that the lazy intruder "generates" all agent names from his initial intruder knowledge IK_0. That is, for symbolic agent names $\mathsf{A}_1, \ldots, \mathsf{A}_p$ in the initial state we have the constraint

$$\mathit{from}(\{\mathsf{A}_1, \ldots, \mathsf{A}_p\}, IK_0) \cup I_0 \quad ,$$

where I_0 is the initial set of inequalities and we assume that IK_0 contains the names of all agents (which is usually not a problem, at least for protocols not involving anonymity or pseudonymity). Note, however, that this constraint actually allows more than we want: each A_j can be instantiated with an arbitrary term that can be generated from IK_0 (e.g. the concatenation of two agent names). We can simply exclude this by enforcing the A_j's to be typed variables (that can only be instantiated with constants of type Agent). Integrating a typed or partially typed model (where only part of the type-information is checked as in this case) into OFMC's untyped model is technically not difficult and can be realized in different ways, e.g. by extending the term algebra with unary functions such as $\mathsf{agent}(\cdot)$.

Regarding the set Agent as a type even allows us to use an infinite set of agent names, with the special rule that the intruder knows every constant of type Agent. Although for a finite number of sessions we only need a finite set of agents (everything else will be equivalent modulo renaming of constant agent names), it can be difficult to determine how many distinct agents are actually necessary (in particular, if there is some form of negation in the model). An infinite type Agent makes matters simpler in this regard. Also, the fact that the intruder knows all agent names

[9] This specification of the initial state is slightly more general than the enumeration of ground sessions described above; there, the state facts of the same session ID contain the same agent names, i.e. it results from unifying $\mathsf{A}_1 = \mathsf{A}_2$, $\mathsf{B}_1 = \mathsf{B}_2$, etc:

$$\begin{array}{c}\mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess1},\mathsf{A}_1,\mathsf{B}_1).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess1},\mathsf{B}_1,\mathsf{A}_1).\\ \mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess2},\mathsf{A}_3,\mathsf{B}_3).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess2},\mathsf{B}_3,\mathsf{A}_3).\\ \mathsf{state}(\mathsf{roleA},\mathsf{step0},\mathsf{sess3},\mathsf{A}_5,\mathsf{B}_5).\mathsf{state}(\mathsf{roleB},\mathsf{step0},\mathsf{sess3},\mathsf{B}_5,\mathsf{A}_5)\\ \mathsf{A}_1 \neq \mathsf{i} \wedge \mathsf{B}_1 \neq \mathsf{i} \wedge \mathsf{A}_3 \neq \mathsf{i} \wedge \mathsf{B}_3 \neq \mathsf{i} \wedge \mathsf{A}_5 \neq \mathsf{i} \wedge \mathsf{B}_5 \neq \mathsf{i} \end{array} \quad .$$

implies that the initial constraint set is always satisfiable, and its ground solutions are exactly the possible instantiations of the sessions. In particular, as an immediate consequence of the definitions, we have:

Lemma 5. *Consider a symbolic initial state with variables* $\mathsf{A}_1, \ldots, \mathsf{A}_p$ *of type* Agent*, and an initial intruder knowledge* IK_0 *such that* $\mathsf{Agent} \subseteq IK_0$*. Then the semantics of the initial constraint set is the set of all substitutions of the variables* A_j *with agent names, i.e.*

$$[\![\mathit{from}(\{\mathsf{A}_1, \ldots, \mathsf{A}_p\}, IK_0)]\!] = \{\, \sigma \mid \mathsf{A}_j\sigma \in \mathsf{Agent} \text{ for } 1 \leq j \leq p \}\,.$$

Hence, the lazy intruder can be straightforwardly adapted to solve the problem of session instantiation.

Example 6. To illustrate session instantiation at work, let us now consider the Needham-Schroeder Public Key Protocol NSPK [21, 36]

```
1. A -> B: {NA,A}KB
2. B -> A: {NA,NB}KA
3. A -> B: {NB}KB
```

which aims at providing mutual authentication between two agents but suffers from the well-known man-in-the-middle attack first reported by Lowe. Let us focus, in particular, on the states that form the trace for the man-in-the-middle attack, in the symbolic model where there is one session. (Indeed, using symbolic sessions we only need one session, i.e. one pair of honest agents, to find the attack.)

The initial state in this case contains

$$\begin{array}{c} \mathsf{state}(\mathsf{roleA}, \mathsf{step0}, \mathsf{sess1}, \mathsf{A}_1, \mathsf{B}_1).\mathsf{state}(\mathsf{roleB}, \mathsf{step0}, \mathsf{sess1}, \mathsf{B}_2, \mathsf{A}_2) \\ \mathsf{A}_1 \neq \mathsf{i} \wedge \mathsf{B}_2 \neq \mathsf{i} \end{array} \quad .$$

The attack-trace starts with the agent A_1 sending a message for B_1, encrypted with B_1's public key. The intruder can analyze this message iff $\mathsf{B}_1 = \mathsf{i}$; hence, we have a case-split, i.e. one state where we perform the substitution $\mathsf{B}_1 = \mathsf{i}$ and one where we have the additional inequality $\mathsf{B}_1 \neq \mathsf{i}$. The attack corresponds to the first case, where the substitution is performed. In this case, the intruder learns the nonce $\mathsf{n1}$ that A_1 has created for him. In the next step, the intruder sends a message to B_2, posing as A_2 (whoever that is). B_2 responds with a message encrypted with the public key of A_2, and again the intruder can decrypt that message iff $\mathsf{A}_2 = \mathsf{i}$. Now consider the case $\mathsf{A}_2 \neq \mathsf{i}$. In this case, the intruder chooses to send (under his real name) an answer to A_1's first message. A_1 expects this message to be encrypted with his public key and to contain the nonce he sent earlier to i, i.e. we have the constraint set

$$\begin{array}{ll} \{ & \mathit{from}(\{\mathsf{A}_1, \mathsf{A}_2, \mathsf{B}_2\}, IK_0)\,, \\ & \mathit{from}(\mathsf{NA}, IK_0 \cup \mathsf{n1})\,, \\ & \mathit{from}(\{\mathsf{n1}, \mathsf{NB}\}_{\mathsf{k}(\mathsf{A}_1)}, IK_0 \cup \mathsf{n1} \cup \{\mathsf{NA}, \mathsf{n2}\}_{\mathsf{k}(\mathsf{A}_2)}) \,\} \end{array}$$

and the inequalities

$$\mathsf{A}_1 \neq \mathsf{i} \wedge \mathsf{B}_2 \neq \mathsf{i}\,.$$

Here NA is a variable representing the nonce the intruder sent earlier to B_2, $\mathsf{n1}$ is the nonce the agent A_1 has created for i, $\mathsf{n2}$ is the nonce the agent B_2 has created for A_2, and NB is a variable for whatever the intruder sends to A_1 as his nonce.

The constraint reduction will identify two possible solutions for this constraint set: either the intruder generated the last message $\{\mathsf{n1}, \mathsf{NB}\}_{\mathsf{k}(\mathsf{A}_1)}$ from its components (which he can do, as he knows A_1, the key-table k, and the nonce $\mathsf{n1}$) or he replays the message he just received from A_2, since it is unifiable using the substitution

$$[\,\mathsf{A}_2 \mapsto \mathsf{A}_1\,,\ \mathsf{NA} \mapsto \mathsf{n1}\,,\ \mathsf{NB} \mapsto \mathsf{n2}\,]\,.$$

Hence, we have found out that this trace (which leads to Lowe's man-in-the-middle attack) only works if the agent that B_2 thinks he is talking to (i.e. A_2) is indeed the one who receives the message (i.e. A_1), while A_1 thinks he is talking to i. As an answer, A_1 sends the final message of the protocol $\{\mathsf{n2}\}_{\mathsf{k(i)}}$, so the intruder knows the nonce that B_2 has generated for $\mathsf{A}_2 \neq \mathsf{i}$, which is a violation of secrecy. Furthermore, authentication is violated if he sends this nonce to B_2. □

This example demonstrates how, during search, the space of possible instances is narrowed down in a demand-driven fashion. Note that in the manual session generation, even for only three agents $\mathsf{Agent} = \{\mathsf{a}, \mathsf{b}, \mathsf{i}\}$, we would have 24 instances. However, under the demand-driven strategy of the lazy intruder, not all of these instances must be explored.

To conclude this section, observe that during the example derivation, variables were instantiated with other variables or with the constant i. In general, the A_j's can be instantiated only with those constant agent names that appear in the initial state or in some transition rule, and this is usually just the name of the intruder i. Note that this bears some similarity to the phenomenon observed in [23], namely that under certain conditions "two agents are sufficient" (an honest agent and a dishonest one). Investigating this relationship in more detail will be subject of future work.

7 Experimental Results

To assess the effectiveness and performance of OFMC, we have tested it on a large protocol suite, which includes the authentication protocols of the Clark/Jacob library [21, 28], as well as a number of industrial-scale protocols. Since OFMC implements a semi-decision procedure, it does not terminate for secure protocols, although it can establish the security of protocols for a bounded number of sessions. We describe below the search times we have measured for finding attacks in flawed protocols.

7.1 The Clark/Jacob Library

OFMC can find all known attacks and discovers a new one in a test-suite of 38 protocols from the Clark/Jacob library. As the performance times in Table 1 show, 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. The time displayed for each attack is the one measured for the minimum number of protocol sessions such that the attack can be performed (which is two sessions in most cases). The experiments were carried out 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, OFMC requires a negligible amount of memory.

To our knowledge, there are only few automated tools, such as [28, 39], that have comparable coverage. Most existing tools have been implemented only as prototypes and have only been applied to a small number of examples. In terms of efficiency, there are also few tools comparable to OFMC. The lazy intruder technique provides a great advantage with respect to more "naïve" techniques. Other tools based on the lazy intruder technique, such as CL-atse [50], have comparable performance although currently smaller 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, we report in Table 1 both attacks found. In all other cases, the times are obtained using the untyped model. "MITM" abbreviates man-in-the-middle attack and "STS" abbreviates replay attack based on a short-term secret. Also note that the table contains four variants of protocols in the library, marked with a "*", that we have additionally analyzed.

7.2 A New Attack on the Yahalom Protocol

The Clark/Jacob library reports an attack on the Yahalom protocol, but this attack requires that the intruder can guess the nonce NB, which is contrary to the usual assumption of unguessability of nonces. Using OFMC, however, we have uncovered the subtle weakness that we presented in

Protocol Name	**Attack**	**Time (s)**
ISO symm. key 1-pass unilateral auth.	Replay	0.0
ISO symm. key 2-pass mutual auth.	Replay	0.0
Andrew Secure RPC prot.	Type flaw Replay	0.0 0.1
ISO CCF 1-pass unilateral auth.	Replay	0.0
ISO CCF 2-pass mutual auth.	Replay	0.0
Needham-Schroeder Conventional Key	STS	0.3
Denning-Sacco (symmetric)	Type flaw	0.0
Otway-Rees	Type flaw	0.0
Wide Mouthed Frog	Parallel-session	0.0
Yahalom	Type flaw	0.0
Woo-Lam Π_1	Type flaw	0.0
Woo-Lam Π_2	Type flaw	0.0
Woo-Lam Π_3	Type flaw	0.0
Woo-Lam Π	Parallel-session	0.2
Woo-Lam Mutual auth.	Parallel-session	0.3
Needham-Schroeder Signature prot.	MITM	0.1
* *Neuman Stubblebine initial part*	Type flaw	0.0
* *Neuman Stubblebine rep. part*	STS	0.0
Neuman Stubblebine (complete)	Type flaw	0.0
Kehne Langendorfer Schoenw. (rep. part)	Parallel-session	0.2
Kao Chow rep. auth., 1	STS	0.5
Kao Chow rep. auth., 2	STS	0.5
Kao Chow rep. auth., 3	STS	0.5
ISO public key 1-pass unilateral auth.	Replay	0.0
ISO public key 2-pass unilateral auth.	Replay	0.0
* *Needham-Schroeder Public Key NSPK*	MITM	0.0
NSPK with key server	MITM	1.1
* *NSPK with Lowe's fix*	Type flaw	0.0
SPLICE/AS auth. prot.	Replay	4.0
Hwang and Chen's modified SPLICE	MITM	0.0
Denning Sacco Key Distr. with Public Key	MITM	0.5
CCITT X.509	Type flaw	0.1
Shamir Rivest Adelman Three Pass prot.	Type flaw	0.0
Encrypted Key Exchange	Parallel-session	0.1
Davis Swick Private Key Certificates (DSPKC), prot. 1	Type flaw Replay	0.1 1.2
DSPKC, prot. 2	Type flaw Replay	0.2 0.9
DSPKC, prot. 3	Replay	0.0
DSPKC, prot. 4	Replay	0.0

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

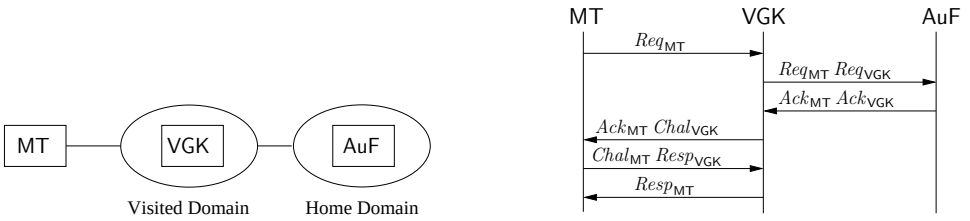

Figure 3: The H.530 protocol (simplified). The deployment of the protocol is on the left, the messages exchange between the participants are summarized on the right.

Fig. 1. In this trace, the intruder officially (under his real name i) plays in the role roleA, and the agents b and s play in the roles roleB and roleS. The notation i(b) denotes that the intruder poses as b. According to the protocol, the intruder receives the new session-key fresh(idKAB, sess2) 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 k(b, s) and starts with the expected agent name i. Hence, b accepts the pairing of nonces NA, fresh(idNB,sess2) as the session-key issued by the server. Although the intruder does not "get in" with this attack (e.g. he did not make the agent b believe that he is talking to somebody else or find out secrets of other sessions), this state violates an authentication goal, namely that any agent playing in the role roleB can rely on the integrity of the session-key. Here, the intruder can make the agent b accept a fake key that did not originate from the server. A part of that key, i's nonce NA, is completely determined by the intruder.

We conclude with three remarks. First, in [45] Paulson proved non-injective agreement (in the sense of [37]) for the Yahalom protocol, including the goal we have found to be violated. However, 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 [21]. There, the intruder listens to the communication between honest agents, and then, similar to our attack, tries to generate message 4 taking advantage of 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 NB). Finally, the attack we have detected is different from the one of [46], in which the intruder only makes the agent playing in the role roleB accept for the second time the key KAB generated by the server. This is a replay attack violating agreement [37]. In our attack, the intruder makes the agent playing in the role roleB accept a key *different* from the one issued by the server.

7.3 The H.530 Protocol

We have applied OFMC to a number of industrial-scale protocols, such as IKE (for which we found the weaknesses already reported in [40]), and the H.530 protocol of the ITU [33]. H.530, which has been developed by Siemens, provides mutual authentication and key agreement in mobile roaming scenarios in multimedia communication.

H.530 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 AuF (denoted by the messages Req_{MT} and Req_{VGK}, respectively). After checking these messages, AuF replies with appropriate acknowledge messages Ack_{MT} and Ack_{VGK} that also contain encrypted hashes for the respective 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 messages exchanged in the H.530 protocol are considerably more complex than the ones

of the Clark/Jacob protocols. As an example, the following excerpt from our HLPSL-specification of H.530 (in ASCII syntax) corresponds to the fourth protocol message

```
4. VGK -> MT : MT,VGK,CH1,CH2,exp(G,Y),
               Hash(K, xor(exp(G,X),exp(G,Y))),Hash(K,MT),
               Hash(exp(exp(G,X),Y),MT,VGK,CH1,CH2,exp(G,Y),
                    Hash(K,xor(exp(G,X),exp(G,Y))),Hash(K,MT))
```

Here, for instance, exp(G, Y) stands for exponentiation, K is a symmetric key, and G, CH1, and CH2 are nonces. In contrast to other model-checking tools, e.g. [28, 38], this kind of complexity is not a problem for our approach. We can directly analyze the full specification of the H.530 without resorting to abstraction or other techniques to simplify the messages.

We have applied OFMC to automatically analyze this protocol in collaboration with Siemens. OFMC takes only 1.6 seconds to detect a previously 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 in the protocol and Siemens has revised the protocol accordingly [34].

8 Related Work and Concluding Remarks

Our formal protocol model is based on the specification languages HLPSL and IF, which we have developed with colleagues as part of a larger project [2, 3, 4, 18, 19, 20, 47] aimed at providing security protocol validation tools. The HLPSL evolved out of the input language of the Casrul system, described in [35].

The use of a generic high-level language and a lower-level language based on (multi)set rewriting was developed by [26] and our work was inspired by this combination. There are, however, a number of differences between our work and the CAPSL/CIL system, proposed in [26]. 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 and conditions, which extends the scope of the protocols and properties that can be modeled. Finally, based on the available experiments (cf. §7 as well as [8, 26]), 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 [28, 38], a compiler that translates protocol specifications, written in a high-level specification 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 successfully discovered flaws in a wide range of protocols: among the protocols of the Clark/Jacob library, it 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. For example, Casper/FDR2 misses our type-flaw attack on Yahalom. Finally, the Casper grammar only includes 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 [49] combines model-checking and interactive theorem-proving techniques with strand spaces [30] to reduce the search space and automatically prove the security of protocols with arbitrary numbers of concurrent runs. Interactive theorem-proving in this setting allows one to limit the search space by manually proving lemmata (e.g. "the intruder cannot find out a certain key, as it is never transmitted"). However, the amount of user interaction necessary to obtain such statements can be considerable. Moreover, like Casper/FDR2, Athena supports only atomic keys and cannot detect type flaws.

To compare with related approaches to symbolically modeling intruder actions, we now expand on the remarks in §4; see [22] for a detailed overview of the lazy intruder approaches. The idea of a symbolic intruder model has undergone a steady evolution, becoming increasingly simpler, more general, and better understood. In the earliest work [32], both the technique itself and the proof were of substantial complexity. [1] drastically simplified the technique and its formal presentation (in particular, the proofs of correctness and completeness), and proved that the constraint reduction problem is NP-hard. [19] presented the first approach that can handle non-atomic keys, albeit without a proof of its correctness and completeness, and [47] showed that, for a bounded number of sessions, the protocol insecurity problem with non-atomic keys is NP-complete. [41] independently proposed a similar generalization to non-atomic keys (although in their case the public key infrastructure is fixed) and gave a simpler proof than [1]. [25] improved the approach of [1] by increasing the expressiveness and providing a more efficient implementation. Note that all approaches that can handle non-atomic keys use the marking of decrypted terms during the analysis of keys as discussed in §4.2 (see the alternative intruder rule $A^l_{\text{scrypt}^*}$). In contrast, our approach, which is closest to the one of [19, 47], works without such a marking, and this provides for a simpler procedure and simpler proofs of its correctness and completeness.

In our work, we have also extended the lazy intruder by introducing inequalities and symbolic sessions. The introduction of inequalities, together with the notion of simple constraints, has a very natural interpretation: "the intruder can generate as many different terms as he likes". As a comparison, inequalities are introduced also in [1], where they are used to handle conditional transitions. The support of inequalities is crucial for a number of advanced protocols and goals, e.g. the replay-check described in Example 1. Our use of inequalities however goes beyond the handling of conditional transitions, as we employ them to handle the entire instantiation problem using only the lazy intruder (while other approaches tackle the instantiation problem by performing an additional parameter search). Moreover, the use of inequalities allows us to explicitly exclude certain substitutions, which is necessary for separating the analysis of messages from constraint reduction, as we discussed in §5.

As we have seen, most approaches are restricted to atomic keys.[10] This prevents the modeling of 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.

The work presented here originated with the idea of on-the-fly model-checking proposed in [7, 8]. 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 the Clark/Jacob protocols, as our example of the H.530 protocol suggests.

Current work involves applying OFMC to other industrial-scale protocols, such as those proposed by the IETF. Although our 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 of reduction techniques inspired by partial-order reduction in our model-checker and the first results are very positive [10].

[10] For example, [12] generalizes previous work [11] by introducing a generic set of cryptographic primitives, but the approach is still limited to atomic keys and it is unclear how this can be lifted without losing the genericity. Note that all cryptographic primitives that are given as examples in [12] are also implemented in OFMC.

References

[1] R. Amadio and D. Lugiez. On the Reachability Problem in Cryptographic Protocols. In *Proceedings of CONCUR'00*, LNCS 1877, pages 380–394. Springer-Verlag, 2002.

[2] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In *Proceedings of CAV'02*, LNCS 2404, pages 349–354. Springer-Verlag, 2002.

[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] A. Armando, L. Compagna, and P. Ganty. SAT-based Model-Checking of Security Protocols using Planning Graph Analysis. In *Proceedings of FME 2003*, LNCS 2805. Springer-Verlag, 2003.

[5] AVISPA: Automated Validation of Internet Security Protocols and Applications. FET Open Project IST-2001-39252, `www.avispa-project.org`.

[6] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[7] D. Basin. Lazy Infinite-State Analysis of Security Protocols. In *Proceedings of CQRE'99*, LNCS 1740, pages 30–42. Springer-Verlag, 1999.

[8] D. Basin and G. Denker. Maude versus Haskell: an Experimental Comparison in Security Protocol Analysis. In *ENTCS 36*. Elsevier, 2001.

[9] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, 2003.

[10] D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. In *Proceedings of CCS'03*, pages 335–344. ACM Press, 2003.

[11] M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proceedings of ICALP'01*, LNCS 2076, pages 667–681. Springer-Verlag, 2001.

[12] M. Boreale and M. G. Buscemi. A Framework for the Analysis of Security Protocols. In *Proceedings of CONCUR'02*, LNCS 2421, pages 483–498. Springer-Verlag, 2002.

[13] M. Boreale and M. G. Buscemi. On the Symbolic Analysis of Low-Level Cryptographic Primitives: Modular Exponentiation and the Diffie-Hellman Protocol. In *Proceedings of MFCS'03*, LNCS 2747. Springer-Verlag, 2003.

[14] M. Bouallagui and H. Jain. Automatic Session Generation. AVISPA report, LORIA-INRIA-Lorraine, Nancy, 2003.

[15] 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.

[16] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with Xor. In *Proceedings of LICS 2003*. IEEE Computer Society Press, 2003.

[17] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proceedings of FSTTCS'03*. Springer, 2003.

[18] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Extending the Dolev-Yao Intruder for Analyzing an Unbounded Number of Sessions. In *Proceedings of CSL 2003*, LNCS 2803. Springer-Verlag, 2003.

[19] Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proceedings of ASE'01*. IEEE Computer Society Press, 2001.

[20] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In *Proceedings of CAV'02*, LNCS 2404, pages 324–337. Springer-Verlag, 2002.

[21] 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`.

[22] H. Comon and V. Shmatikov. Is It Possible to Decide Whether a Cryptographic Protocol Is Secure Or Not? *Journal of Telecommunications and Information Technology*, 4:5–15, 2002.

[23] H. Comon-Lundh and V. Cortier. Security Properties: Two Agents are Sufficient. In *Proceedings of ESOP'03*, LNCS 2618, pages 99–113. Springer-Verlag, 2003.

[24] H. Comon-Lundh and V. Shmatikov. Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive Or. In *Proceedings of LICS 2003*. IEEE Computer Society Press, 2003.

[25] 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.

[26] 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.

[27] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[28] 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.

[29] 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.

[30] 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.

[31] M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proceedings of CSFW'01*. IEEE Computer Society Press, 2001.

[32] 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.

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

[34] ITU-T Recommendation H.530, Corrigendum 1. 2003. Corrected version of [33].

[35] 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.

[36] G. Lowe. Breaking and Fixing the Needham-Shroeder Public-Key Protocol Using FDR. In *Proceedings of TACAS'96*, LNCS 1055, pages 147–166. Springer-Verlag, 1996.

[37] G. Lowe. A Hierarchy of Authentication Specifications. In *Proceedings of CSFW'97*, pages 31–43. IEEE Computer Society Press, 1997.

[38] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[39] C. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[40] 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.

[41] J. K. Millen and V. Shmatikov. Constraint Solving for Bounded-Process Cryptographic Protocol Analysis. In *Proceedings of CCS'01*, pages 166–175. ACM Press, 2001.

[42] J. K. Millen and V. Shmatikov. Symbolic Protocol Analysis with Products and Diffie-Hellman Exponentiation. In *Proceedings of CSFW'03*. IEEE Computer Society Press, 2003.

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

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

[45] 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.

[46] 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.

[47] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *Proceedings of CSFW'01*. IEEE Computer Society Press, 2001.

[48] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2000.

[49] 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.

[50] M. Turuani. *Sécurité des Protocoles Cryptographiques: Décidabilité et Complexité*. Phd thesis, Université Henri Poincaré, Nancy, 2003.

A Proofs of the Theorems and Lemmata

We now give the proofs of the theorems and lemmata stated in the body of the paper. We will employ the following additional notation.

Notation 2. Given a constraint $from(T, IK)$, we use LHS and RHS to refer to T and IK, respectively.

Our first theorem, Theorem 1, follows directly from Lemmata 6, 7, and 8.

Theorem 1 *Let C be a well-formed constraint set. $Red(C)$ is finite and $\vdash$ is well-founded. Moreover, $[\![C]\!] = [\![Red(C)]\!]$, i.e. $Red(C)$ is correct and complete.*

Lemma 6. *Let C be a well-formed constraint set. $Red(C)$ is finite and $\vdash$ is well-founded.*

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 only be 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 the other generation rules of the lazy intruder decompose LHS terms. The applicability of analysis rules is bounded because only subterms of the initial RHS terms can occur in the resulting intruder knowledge. Formally:

Proof. Let us begin by defining a *weight function w for messages on the LHS of a constraint* as follows:

$$\begin{aligned}
w(m) &= 1, \quad \text{for } m \in \mathit{AtomicMsg} \\
w(m^{-1}) &= 1 \\
w(\langle m_1,m_2\rangle) &= w(m_1) + w(m_2) + 1 \\
w(\{|m_2|\}_{m_1}) &= w(m_1) + w(m_2) + 1 \\
w(\{m_2\}_{m_1}) &= w(m_1) + w(m_2) + 1 \\
w(m_1(m_2)) &= w(m_1) + w(m_2) + 1 \\
w(\{m_1,\ldots,m_n\}) &= \sum_{i=1}^{n} w(m_i)
\end{aligned}$$

The definition of the weight of the intruder knowledge (i.e. of the RHS of the constraints) must take into account that analysis is possible that introduces a new term into the intruder knowledge as well as a new constraint for the derivation of the key. To measure the weight of the intruder knowledge, we thus define the *weight function w_{IK} for messages in the IK in the RHS of a constraints* as follows:

$$\begin{aligned}
w_{IK}(m) &= w(m), \quad \text{for } m \in \mathit{AtomicMsg} \\
w_{IK}(m^{-1}) &= w(m^{-1}) \\
w_{IK}(\langle m_1,m_2\rangle) &= \begin{cases} w(\langle m_1,m_2\rangle), \text{ if } m_1 \in IK \text{ and } m_2 \in IK \\ w(\langle m_1,m_2\rangle) + w_{m_1 \cup m_2 \cup IK}(m_1 \cup m_2) + 1, \\ \qquad \text{if } m_1 \notin IK \text{ or } m_2 \notin IK \end{cases} \\
w_{IK}(\{|m_2|\}_{m_1}) &= \begin{cases} w(\{|m_2|\}_{m_1}), \text{ if } m_2 \in IK \\ w(\{|m_2|\}_{m_1}) + w_{m_2 \cup IK}(m_2) + w(m_1) \\ \qquad +w_{IK'}(IK') + 1, \text{ for } IK' = IK \setminus \{|m_2|\}_{m_1}, \\ \qquad \text{if } m_2 \notin IK \end{cases} \\
w_{IK}(\{m_2\}_{m_1}) &= \begin{cases} w(\{m_2\}_{m_1}), \text{ if } m_2 \in IK \\ w(\{m_2\}_{m_1}) + w_{m_2 \cup IK}(m_2) + w(m_1) \\ \qquad +w_{IK'}(IK') + 1, \text{ for } IK' = IK \setminus \{m_2\}_{m_1}, \\ \qquad \text{if } m_2 \notin IK \end{cases} \\
w_{IK}(m_1(m_2)) &= w(m_1(m_2)) \\
w_{IK}(\{m_1,\ldots,m_n\}) &= \sum_{i=1}^{n} w_{IK}(m_i)
\end{aligned}$$

Observe now that if $IK' \subseteq IK$ then $w_{IK'}(M) \geq w_{IK}(M)$ for all sets M of messages. This holds as the only places in w_{IK} that depend on the set IK are in the case-splits for the cases $\langle m_1,m_2\rangle$,

$\{\!|m_2|\!\}_{m_1}$, and $\{m_2\}_{m_1}$. In these cases, a smaller IK can only lead to a larger weight as the second cases of the case-splits add in additional weights.

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

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

We order these pairs lexicographically, overloading the ">" symbol, defining

$$(v_1, g_1) > (v_2, g_2) \ \text{ iff } \ v_1 > v_2\ \vee\ (v_1 = v_2\ \wedge\ g_1 > g_2)\,.$$

This order is well-founded, as the ordering on both components is well-founded 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}: if no variable is substituted, then a term from the LHS is dropped (so strictly less in g), else at least one variable is substituted (so strictly less in v).
- G^l_{pair}, G^l_{scrypt}, G^l_{crypt}, and G^l_{apply}: 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_{crypt}, $A^l_{\mathrm{crypt}^{-1}}$, and 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,
$$\sum_{i=1}^{n} w(T_i)\ +\ w_{IK_i}(IK_i)\,.$$
If we apply an analysis rule to the jth constraint, $1 \le j \le n$, and $IK_j = \{t_1, \ldots, t_p\}$ and the analyzed message is $t_l = \{\!|m_2|\!\}_{m_1}$ for $l \in \{1, \ldots, p\}$, then the constraint after application of the analysis rule is
$$\{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_{j-1}, IK_{j-1}),\ \mathit{from}(m_1, IK_j \backslash \{t_l\}),\ \mathit{from}(T_j, m_2\ \cup\ IK_j), \mathit{from}(T_{j+1}, IK_{j+1}), \ldots, \mathit{from}(T_n, IK_n)\}$$
with the weight (second component):
$$\begin{array}{lll} & O + w(m_1) + w_{IK_0}(IK_0) + w(T_j) + w_{m_2 \cup IK_j}(m_2\ \cup\ IK_j) \\ = & O + w(m_1) + w_{IK_0}(IK_0) + w(T_j) + w_{m_2 \cup IK_j}(IK_0) \\ & +w(\{\!|m_2|\!\}_{m_1}) + w_{m_2 \cup IK_j}(m_2) \\ \le & O + w(m_1) + w_{IK_0}(IK_0) + w(T_j) + w_{IK_j}(IK_0) + w(\{\!|m_2|\!\}_{m_1}) \\ & +w_{m_2 \cup IK_j}(m_2) \\ =_{\mathrm{def}} & O + w(T_j) + w_{IK_j}(IK_0) + w_{IK_j}(\{\!|m_2|\!\}_{m_1}) - 1 \\ = & O + w(T_j) + w_{IK_j}(IK_j) - 1 \\ = & \mathit{snd}(W(\{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_n, IK_n)\}) - 1\,, \end{array}$$
where $O = \mathit{snd}(W(\{\mathit{from}(T_1, IK_1), \ldots, \mathit{from}(T_n, IK_n)\} \setminus \mathit{from}(T_j, IK_j)))$ and $IK_0 = IK_j \setminus \{\!|m_2|\!\}_{m_1}$. So the analysis decreases the weight by at least one. This concludes the proof of the lemma.

□

Lemma 7. $[\![C]\!] \supseteq [\![Red(C)]\!]$ *for a well-formed constraint set* C.

Proof. Let C be a well-formed constraint set. Initially, observe that $[\![C]\!] \supseteq [\![Red(C)]\!]$ abbreviates $[\![C]\!] \supseteq \{\sigma\sigma' \mid \exists C'.\ (C', \sigma) \in Red(C) \wedge \sigma' \in [\![C']\!]\}$. To show 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 is correct in the following sense. Whenever the application of a rule transforms a constraint set and a substitution (C, τ) into $(C', \tau\sigma)$, then any solution for C' is also a solution for C, i.e. $\sigma' \in [\![C']\!] \implies \sigma\sigma' \in [\![C]\!]$ (where τ is not considered as $\mathrm{vars}(C) \cap \mathrm{dom}(\tau) = \emptyset$ by the construction of *Red*). 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 $\sigma' \in [\![C']\!]$. Then σ' already satisfies all constraints in C except $\mathit{from}(t \cup T, s \cup IK)$. Now $t\sigma\sigma' = s\sigma\sigma'$ and hence $\sigma\sigma'$ is a solution of C. □

Lemma 8. $[\![C]\!] \subseteq [\![Red(C)]\!]$ *for a well-formed constraint set* C.

Proof. Let C be a well-formed constraint set. We begin the proof by observing that $[\![C]\!] \subseteq [\![Red(C)]\!]$ abbreviates $[\![C]\!] \subseteq \{\sigma\sigma' \mid \exists C'.\ (C', \sigma) \in Red(C) \wedge \sigma' \in [\![C']\!]\}$. Showing the completeness of *Red*, i.e. that all solutions of a constraint set C are also solutions of the reduction of C, is more difficult than showing the correctness of *Red*. The main problem is that completeness requires that *Red* is performed on a well-formed constraint set, but, during the reduction procedure, property (10) of the well-formedness (cf. Definition 9) can be destroyed by the analysis rules. In particular, 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, "previous" and "successive" constraints refer to the order given by the well-formed constraint sets.

Problem (ii) is easy to overcome. To restore property (10) we perform the same analysis steps also on the successive constraints: these must allow the same derivations in the intruder knowledge since 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 the application of an analysis rule. For simplicity, we will still refer to this invariant as well-formedness. For the resulting simple constraint sets, one can restore their well-formedness in the original sense by simply deleting the constraints that were introduced by applications of analysis rules.[11]

We say in the following that *the result* (C', σ) *of a reduction step* (or, simply, *reduction*) *supports* τ if there exists a σ' such that $\tau = \sigma\sigma'$ and $\sigma' \in [\![C']\!]$. Hence, our proof obligation is to show that for a given well-formed constraint C and an arbitrary solution $\tau \in [\![C]\!]$, the reduction yields at least one result $(C', \sigma) \in Red(C)$ 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 (C', σ) of the rule application supports τ. Once this property of the rules is proved, we show 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. By Lemma 6, 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

[11] This does not change the semantics of the constraint set. The constraint set is simple and therefore all constraints have only variables in 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 in 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.

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 this derivation explicit in the constraint set C by labeling 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. Note that we will simplify the notation by writing only the messages and omitting the "$\in \mathcal{DY}(IK)$"; in particular, the applications of the G_{axiom} rule will simply be the leaf nodes of the tree.

Every leaf and every node then stands for a message, composed or decomposed from the (roots of 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 labeled 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 labeled with a $\mathcal{DY}$-derivation tree of $t\tau$ from terms in $IK\tau$.

As a simple example, consider $C = \mathit{from}(\langle \mathsf{K},\mathsf{m}\rangle, \{\!|\mathsf{m}|\!\}_{\mathsf{k}} \cup \{\!|\mathsf{k}|\!\}_{\{\!|\mathsf{m}|\!\}_{\mathsf{k}}})$, which has, among others, the solution $\tau = [\mathsf{K} \mapsto \{\!|\mathsf{m}|\!\}_{\mathsf{k}}]$. We can label the message $\langle \mathsf{K},\mathsf{m}\rangle$ with the derivation tree for the message $\langle \mathsf{K},\mathsf{m}\rangle\tau = \langle \{\!|\mathsf{m}|\!\}_{\mathsf{k}},\mathsf{m}\rangle$ as follows:

$$\dfrac{\dfrac{\dfrac{}{\{\!|\mathsf{m}|\!\}_{\mathsf{k}}}\,G_{\text{axiom}} \quad \dfrac{}{\{\!|\mathsf{k}|\!\}_{\{\!|\mathsf{m}|\!\}_{\mathsf{k}}}}\,G_{\text{axiom}}}{\mathsf{k}}\,A_{\text{scrypt}} \quad \dfrac{}{\{\!|\mathsf{m}|\!\}_{\mathsf{k}}}\,G_{\text{axiom}}}{\mathsf{m}}\,A_{\text{scrypt}}}{\langle \{\!|\mathsf{m}|\!\}_{\mathsf{k}},\mathsf{m}\rangle}\,G_{\text{pair}}$$
$$\bigtriangledown$$
$$\langle \mathsf{K},\mathsf{m}\rangle \quad .$$

Note that we use the symbol $\bigtriangledown$ to denote labeling of terms with Dolev-Yao derivation trees.

To resume the proof, let a well-formed, non-simple constraint set C and a solution $\tau \in [\![C]\!]$ be given, where C is labeled with $\mathcal{DY}$-derivation trees for the solution τ. According to the order of the well-formed constraint set, 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 that labels t, we can find a constraint reduction rule that is applicable and such that the resulting constraint set C' can again be labeled with $\mathcal{DY}$ trees according to τ (and hence the result still supports τ). We now consider the different possible cases for the kind of root node that the $\mathcal{DY}$-tree has for t.

G_{axiom}: This means that $t\tau \in IK\tau$. So t can be unified with a term $s \in IK$ and the G^l_{unif} rule is applicable to C, and the unifier $\sigma = \mathit{mgu}(t, s)$ is compatible with τ. Hence the resulting C' supports τ as all remaining terms can be labeled with the same trees as in C. To illustrate this, observe that applying the rule G^l_{unif} (using a substitution σ that is at least as general as τ) to the constraint

$$\mathit{from}\left(\begin{array}{c}\dfrac{}{t\tau}\,G_{\text{axiom}} \\ \bigtriangledown \\ t\end{array} \quad \begin{array}{c} T_0 \\ \vdots \\ \cup\, E_0, IK\end{array}\right)$$

yields

$$\begin{array}{c} T_0 \\ \vdots \\ \mathit{from}(E_0\sigma, IK\sigma)\ .\end{array}$$

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

that applying the rule G^l_{scrypt} to the constraint

$$\textit{from}\Big(\ \dfrac{\dfrac{\begin{matrix}T_1\\ \vdots\\ t_1\tau\end{matrix}\quad\begin{matrix}T_2\\ \vdots\\ t_2\tau\end{matrix}}{\{\!|t_2|\!\}_{t_1}\tau}\ G_{\text{scrypt}}}{\{\!|t_2|\!\}_{t_1}}\ \cup\ \begin{matrix}T_0\\ \vdots\\ E_0\end{matrix}, IK\Big)$$

yields

$$\textit{from}\Big(\ \begin{matrix}T_1\\ \vdots\\ t_1\tau\end{matrix}\ \cup\ \begin{matrix}T_2\\ \vdots\\ t_2\tau\end{matrix}\ \cup\ \begin{matrix}T_0\\ \vdots\\ E_0\sigma\end{matrix}, IK\sigma\Big)\ .$$

A_{scrypt} (and similarly for A_{pair}, A_{crypt}, and A^{-1}_{crypt}, mutatis mutandis): $t\tau$ is obtained through a decryption of a term $\{\!|t|\!\}_k\tau$. 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 analysis operations in its subtrees. We pick such an analysis operation for applying the next lazy intruder rule to the constraint set. Let t'_0 be the term obtained in the $\mathcal{DY}$-tree by the selected analysis operation, and k'_0 and $\{\!|t'_0|\!\}_{k'_0}$ be the terms obtained at the children nodes of the analysis operation (in the case that there were no analysis nodes in the subtrees, we have $t'_0 = t\tau$ and $k'_0 = k\tau$). There must be a message $tk_0 \in IK$ such that $tk_0\tau = \{\!|t'_0|\!\}_{k'_0}$. Without loss of generality, we can assume that tk_0 is not a variable: if it were a variable, then, since C is well-formed, $tk_0 \in T$ for an earlier constraint $\textit{from}(T, IK_0) \in C$ (which is already simple) for some $IK_0 \subseteq IK$ with $tk_0 \notin IK_0$. Therefore, the term to be constructed can be generated from the knowledge IK_0. Since tk_0 is not a variable, it must have the form $tk_0 = \{\!|k_0|\!\}_{t_0}$, where $t_0\tau = t'_0$ and $k_0\tau = k'_0$, and therefore the rule A^l_{scrypt} can be applied. This adds the analyzed term t_0 to the intruder knowledge and the new constraint $\textit{from}(k_0, IK \setminus \{\!|t_0|\!\}_{k_0})$ to the constraint set (in the case of A_{pair}, no new constraint is added). Now all occurrences of the analyzed term $t_0\tau$ in the $\mathcal{DY}$-tree can be replaced with a leaf-node. In the newly added constraint $\textit{from}(k_0, IK \setminus \{\!|t_0|\!\}_{k_0})$ (in the case of A_{scrypt}), the term k_0 can be labeled with the $k_0\tau$-subtree of the analysis node in the $\mathcal{DY}$-tree. This labeling is correct since this subtree cannot contain any further analysis operations; therefore it can contain only subterms of $k_0\tau$ (if $k_0\tau$ is non-atomic) that are present in $IK\tau$. In particular it cannot contain $\{\!|t_0|\!\}_{k_0}\tau$. Hence, C' still supports τ. To illustrate this, observe that applying the rule A^l_{scrypt} to the constraint

$$\textit{from}\Big(\ \dfrac{\dfrac{\dfrac{\begin{matrix}T_1\\ \vdots\\ k_0\tau\end{matrix}\quad\dfrac{}{\{\!|t_0|\!\}_{k_0}\tau}\ G_{\text{axiom}}}{t_0\tau}\ A_{\text{scrypt}}\quad\begin{matrix}T_2\\ \vdots\\ \{\!|t|\!\}_k\tau\end{matrix}}{k\tau}\ A_{\text{scrypt}}}{t\tau}\ \cup\ \begin{matrix}T_0\\ \vdots\\ E_0\end{matrix}, IK\Big)$$

yields

$$\textit{from}\Big(\ \begin{matrix}T_1\\ \vdots\\ k_0\tau\end{matrix}, IK \setminus \{\!|t_0|\!\}_{k_0}\Big)\ \cup\ \textit{from}\Big(\ \dfrac{\dfrac{\dfrac{}{t_0\tau}\ G_{\text{axiom}}\quad\begin{matrix}T_2\\ \vdots\\ \{\!|t|\!\}_k\tau\end{matrix}}{k\tau}\ A_{\text{scrypt}}}{t\tau}\ \cup\ \begin{matrix}T_0\\ \vdots\\ E_0\end{matrix}, t_0 \cup IK\Big)\ .$$

This concludes the proof of the lemma. □

To illustrate the proof, we consider again the 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} rule to obtain $\mathit{from}(\mathsf{K} \cup \mathsf{m}, \{\!|\mathsf{m}|\!\}_\mathsf{k} \cup \{\!|\mathsf{k}|\!\}_{\{\!|\mathsf{m}|\!\}_\mathsf{k}})$ labeled as follows (recall we consider the solution $\tau = [\mathsf{K} \mapsto \{\!|\mathsf{m}|\!\}_\mathsf{k}]$:

$$\dfrac{}{\{\!|\mathsf{m}|\!\}_\mathsf{k}}\, G_{\text{axiom}} \qquad \dfrac{\dfrac{\dfrac{}{\{\!|\mathsf{m}|\!\}_\mathsf{k}}\, G_{\text{axiom}} \quad \dfrac{}{\{\!|\mathsf{k}|\!\}_{\{\!|\mathsf{m}|\!\}_\mathsf{k}}}\, G_{\text{axiom}}}{\mathsf{k}}\, A_{\text{scrypt}} \quad \dfrac{}{\{\!|\mathsf{m}|\!\}_\mathsf{k}}\, G_{\text{axiom}}}{\mathsf{m}}\, A_{\text{scrypt}}$$

$$\begin{array}{c} \{\!|\mathsf{m}|\!\}_\mathsf{k} \\ \triangledown \\ \mathsf{K} \end{array} \qquad\qquad \begin{array}{c} \mathsf{m} \\ \triangledown \\ \mathsf{m} \end{array} \quad .$$

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 $\{\!|\mathsf{k}|\!\}_{\{\!|\mathsf{m}|\!\}_\mathsf{k}}$, adds k to the intruder knowledge and adds the new constraint that the key term, $\{\!|\mathsf{m}|\!\}_\mathsf{k}$, can be derived from the rest of the knowledge, i.e. $\mathit{from}(\{\!|\mathsf{m}|\!\}_\mathsf{k}, \{\!|\mathsf{m}|\!\}_\mathsf{k})$, $\mathit{from}(\mathsf{K} \cup \mathsf{m}, \mathsf{k} \cup \{\!|\mathsf{m}|\!\}_\mathsf{k} \cup \{\!|\mathsf{k}|\!\}_{\{\!|\mathsf{m}|\!\}_\mathsf{k}})$, with the labeling

$$\dfrac{}{\{\!|\mathsf{m}|\!\}_\mathsf{k}}\, G_{\text{axiom}} \qquad \dfrac{}{\{\!|\mathsf{m}|\!\}_\mathsf{k}}\, G_{\text{axiom}} \qquad \dfrac{\dfrac{}{\mathsf{k}}\, G_{\text{axiom}} \quad \dfrac{}{\{\!|\mathsf{m}|\!\}_\mathsf{k}}\, G_{\text{axiom}}}{\mathsf{m}}\, A_{\text{scrypt}}$$

$$\begin{array}{c} \{\!|\mathsf{m}|\!\}_\mathsf{k} \\ \triangledown \\ \mathsf{K} \end{array} \qquad \begin{array}{c} \{\!|\mathsf{m}|\!\}_\mathsf{k} \\ \triangledown \\ \{\!|\mathsf{m}|\!\}_\mathsf{k} \end{array} \qquad \begin{array}{c} \mathsf{m} \\ \triangledown \\ \mathsf{m} \end{array} \quad .$$

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

$$\dfrac{}{\{\!|\mathsf{m}|\!\}_\mathsf{k}}\, G_{\text{axiom}} \qquad \dfrac{}{\mathsf{k}}\, G_{\text{axiom}} \qquad \dfrac{}{\mathsf{m}}\, G_{\text{axiom}}$$

$$\begin{array}{c} \{\!|\mathsf{m}|\!\}_\mathsf{k} \\ \triangledown \\ \mathsf{K} \end{array} \qquad \begin{array}{c} \mathsf{k} \\ \triangledown \\ \mathsf{k} \end{array} \qquad \begin{array}{c} \mathsf{m} \\ \triangledown \\ \mathsf{m} \end{array} \quad .$$

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

We now prove Lemma 1, i.e. that the lazy model is equivalent to the ground model, in the sense that they represent the same set of reachable states. To do that, we first prove the following auxiliary lemmata:

Lemma 9. *For a rule $r = lhs \Rightarrow rhs$ of the form* (1) *and a lazy state (P, C, N) where* $\mathrm{vars}(P, C, N) \cap \mathrm{vars}(lhs) = \emptyset$*, it holds that:*

$$\begin{aligned} &\{\sigma\tau \mid \exists S.\ S \in [\![(P, C, N)]\!] \wedge S = P\sigma \wedge \tau \in \mathit{applicable}_{lhs}(S)\} \\ &= \{\sigma\tau \mid \exists C', N'.\ (\sigma, C', N') \in \mathit{applicable}^l_{lhs}(P, C, N) \wedge \\ &\qquad \mathrm{ground}(lhs\sigma\tau) \wedge \mathrm{ground}((P, C, N)\sigma\tau) \wedge \sigma\tau \in [\![C']\!] \wedge \sigma\tau \models N' \}\ . \end{aligned}$$

Proof. For the $\subseteq$ direction, let (P, C, N) be a lazy state and $S \in [\![(P, C, N)]\!]$ be a ground state. Let σ' be the respective ground substitution, i.e. such that $S = P\sigma'$, $\sigma' \in [\![C]\!]$ and $\sigma' \models N$. Let $\tau' \in \mathit{applicable}_{lhs}(S)$. We show for $\sigma = \sigma'\tau'$ and $\tau = \mathrm{id}$ that the conditions of the right-hand side of the equality of the lemma are satisfied, and therefore $\sigma\tau = \sigma'\tau'$ is contained in the right-hand side set.

First, we construct C' and N' for σ according to the definition of applicable, i.e.

$$\begin{aligned} C' &= (C \cup \mathit{from}(m_1 \cup \{m \mid \mathsf{i_knows}(m) \in P_1\}, \{i \mid \mathsf{i_knows}(i) \in P\}))\sigma \\ N' &= N\sigma \wedge \bigwedge_{\phi \in subCont(N_1\sigma, P\sigma)} \phi \wedge \mathit{Cond}\,\sigma\,. \end{aligned}$$

We then have that $(\sigma, C', N') \in \mathit{applicable}^l_{lhs}(P, C, N)$, because the conditions of $\mathit{applicable}^l$ are satisfied: since $\mathsf{state}(m_2\tau') \in S$, $m_2\tau' = m_2\sigma$ and $S = P\sigma' = P\sigma$, it follows that $\mathsf{state}(m_2\sigma) \in P\sigma$; similarly, $\overline{P_1}\sigma \subseteq P\sigma$.

We now show that $\sigma \in [\![C']\!]$. As $\sigma \in [\![C]\!]$, we must only show that the newly added constraint is satisfied by σ, i.e. that $\sigma \in [\![\mathit{from}(m_1 \cup \{m \mid \mathsf{i_knows}(m) \in P_1\}, \{i \mid \mathsf{i_knows}(i) \in P\})]\!]$. The assumption that $\sigma' \in \mathit{applicable}_{lhs}(S)$ implies that the messages $m_1\sigma' \cup \{m\sigma' \mid \mathsf{i_knows}(m) \in P_1\}$ can be generated according to $\mathcal{DY}$ from the intruder knowledge of S, i.e. from $\{m \mid \mathsf{i_knows}(m) \in S\}$. For the messages m of *lhs*, it holds that $m\sigma = m\sigma'$, and the intruder knowledge of S is equal to the one under $P\sigma$. Therefore the constraint is satisfied.

Finally, we show that $\sigma \models N'$. Assume that $\sigma \not\models N'$. Since $\sigma \models N$, and $\sigma \models \mathit{Cond}$, the only possible reason for $\sigma \not\models N'$ is that (at least) one of the conjuncts of $\mathit{subCont}(N_1\sigma, P\sigma)$ does not hold. That would mean that there are a fact $\mathsf{not}(f) \in N_1\sigma$ and a substitution ρ such that $f\sigma\rho \in P\sigma\rho$. Since $f\sigma$ and $P\sigma$ are ground, $f\sigma \in P\sigma$. Since $f\sigma = f\tau'$ and $P\sigma = S$, this contradicts the assumption of $\tau' \in \mathit{applicable}_{lhs}(S)$, which implies $(\forall f.\ \mathsf{not}(f) \in N_1 \implies f\tau' \notin S)$.

The converse direction $\supseteq$ follows similarly, and we therefore only give the basic structure of the proof. Let (P, C, N), σ, τ, C' and N' be as in the conditions of the set comprehension of the right-hand side of the equality. Let σ' be the restriction of $\sigma\tau$ to the variables of (P, C, N) and let τ' be the restriction of $\sigma\tau$ to the variables of *lhs* (since $\sigma\tau$ is a ground substitution for variables of both (P, C, N) and *lhs*). It is then easy to show that $S = P\sigma \in [\![(P, C, N)]\!]$ and $\tau' \in \mathit{applicable}_{lhs}(S)$, which concludes the proof. □

By employing the same construction of substitutions as in the proof of lemma 9, we can straightforwardly show:

Lemma 10. *For a rule r of the form* (1) *and a lazy state (P, C, N) where* $\mathrm{vars}(P, C, N) \cap \mathrm{vars}(r) = \emptyset$ *it holds that:*

$$\bigcup_{S \in [\![(P,C,N)]\!]} \mathit{step}_r(S) = \bigcup_{(P',C',N') \in \mathit{step}^l_r(P,C,N)} [\![(P', C', N')]\!]\,.$$

The main lemma about the reachability in the ground and lazy models follows by induction on the number of applications of *step* and step^l, using Lemma 10, where variable clashes between lazy states and rules are prevented by the renaming performed as part of the succ^l function.

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

Before proving Theorem 2, we show the relationship between ground and lazy attack-predicates:

Lemma 2 *For all lazy states (P, C, N) and all attack-rules ar, the predicate $\mathit{isAttack}^l_{ar}(P, C, N)$ holds iff $\mathit{isAttack}_{ar}(S)$ holds for some represented ground state $S \in [\![(P, C, N)]\!]$.*

Proof. The lemma follows by observing that $\mathit{isAttack}^l_{ar}(P, C, N)$ holds iff there is $(\sigma, C', N') \in \mathit{applicable}^l_{ar}(P, C, N)$ such that $\tau \models N'$ holds for some ground substitution $\tau \in [\![C']\!]$. This is in turn the case iff there is a ground state $S \in [\![(P, C, N)]\!]$ such that $\mathit{applicable}_{ar}(S)$ holds, which follows by Lemma 9. □

Theorem 2 *A protocol (I, R, AR) is* secure *iff $\mathit{isAttack}^l_{ar}(P, C, N)$ is false for all attack-rules $ar \in AR$ and all reachable lazy states $(P, C, N) \in \mathit{reach}^l(I, R)$.*

Proof. Let (I, R, AR) be a secure protocol. This is equivalent to the condition that $isAttack_{ar}(S)$ does not hold for any $ar \in AR$ and reachable state S. By Lemma 1, this is in turn equivalent to the condition that $isAttack_{ar}(S)$ does not hold for any state S in the semantics of a reachable lazy state (P, C, N) and $ar \in AR$. Finally, by Lemma 2, this is equivalent to the condition that the predicate $isAttack^l_{ar}(S)$ does not hold for all reachable lazy states (P, C, N) and $ar \in AR$. □

We conclude by proving Lemma 3 and Lemma 4.

Lemma 3 *Let (P, C, N) be a lazy state where C is simple and N is satisfiable, i.e. there is a σ such that $\sigma \models N$. 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 instances 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.

The key idea behind the integration of inequalities is that, unless the inequalities alone are already unsatisfiable, they cannot destroy the satisfiability of the constraint set, as we now show. It is straightforward to check whether a conjunction of disjunctions of inequalities N is satisfiable. Let $\mathrm{vars}(N) = \{v_1, \ldots, v_n\}$ 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 the variables in N that do not occur in C) is also a solution for N, provided it 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 $\mathrm{vars}(N) = \{v_1, \ldots, v_n\}$ 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 for the untyped model. In the case of the typed model, the proof proceeds along the same lines since we can assume, without loss of generality, that the intruder initially knows (or can freshly create) an unbounded number of messages of each type. □

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

Proof. The lemma follows from the following stronger invariant on the states that can be reached with $step^l$: all reachable states (P, C, N) have the property that

$$C \text{ is well-formed, } \mathrm{vars}(P) \subseteq \mathrm{vars}(C), \text{ and } ik(C) \subseteq ik(P)\,, \tag{18}$$

where we introduce the function ik to denote the *intruder knowledge* for several types of arguments: $ik(P) = \{m \mid \mathsf{msg}(m) \in P \vee \mathsf{i_knows}(m) \in P\}$ for a set P of facts, $ik(from(T, IK)) = IK$ for a constraint, $ik(C) = \cup_{c \in C} ik(c)$ for a set of constraints, $ik(r) = ik(P)$ for a rule r with set P of positive facts in the left-hand side.

Let (P, C, N) be a state that obeys (18), let $(P', C', N') \in step^l_r(P, C, N)$ for some rule $r \in R$, and let σ be the corresponding substitution according to the definition of $step^l$. Note that we can safely assume that $\mathrm{vars}(r) \cap \mathrm{vars}(P', C', N') = \emptyset$ as $succ^l$ renames all rule variables. We show that (P', C', N') obeys (18).

First, observe that in the definition of $step^l$ all the facts of $\overline{P_1}$ (i.e. the positive left-hand side facts of the rule without the intruder-generated facts $ik(r)$) are unified under σ with facts of P. Hence, σ substitutes all those variables of the rule that occur in $\overline{P_1}$. Due to the form of the rules, all other variables of the rule r are those variables that occur in $ik(r)$ but nowhere else in the positive facts of the rule. Let us denote this set of variables with IV. We can thus conclude that $\mathrm{vars}(P') \subseteq \mathrm{vars}(P) \cup IV$. Furthermore, the constraint set is augmented by the constraint

$c = \mathit{from}(ik(r), ik(P))$, i.e. $C' = C \cup \{c\}$. Hence, $\mathrm{vars}(C') = \mathrm{vars}(C) \cup IV \supseteq \mathrm{vars}(P')$, proving the second conjunct of (18) for (P', C', N'). Moreover, for the new constraint c, it holds that $ik(c) = ik(P) \supseteq ik(C)$. It follows that $ik(P') \supseteq ik(P) = ik(c) = ik(C')$, proving the third conjunct of (18) for (P', C', N').

Finally, as C is already well-formed, there is an order on the constraints in C along which the intruder knowledge increases and all variables are introduced on the left-hand sides of the constraints. If we extend such an order making c' the highest constraint in C', then these two properties still hold so that C' is also well-formed. The intruder knowledge increases since, as noted above, $ik(c) \supseteq ik(C)$ and $\mathrm{vars}(c') \setminus \mathrm{vars}(C) = IV = \mathrm{vars}(ik(r)) \setminus \mathrm{vars}(ik(P))$. This proves the first conjunct of (18) for (P', C', N'), and the lemma follows. □