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

David Basin, Sebastian Mödersheim, and Luca Viganò

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

Abstract. We introduce the on-the-fly model-checker OFMC, a tool that combines two methods for analyzing security protocols. The first is the use of lazy data-types as a simple way of building an efficient on-the-fly model checker for protocols with infinite state spaces. The second is the integration of symbolic techniques for modeling a Dolev-Yao intruder, whose actions are generated in a demand-driven way. We present experiments that demonstrate that our tool is state-of-the-art, both in terms of coverage and performance, and that it scales well to industrial-strength protocols.

1 Introduction

A wide variety of model-checking approaches have recently been applied to analyzing security protocols, e.g. [1, 7, 13, 22, 23, 25, 26]. The key challenge they face is that the general security problem is undecidable [14], and even semi-algorithms, focused on falsification, must come to terms with the enormous branching factor in the search space resulting from using the standard Dolev-Yao intruder model, where the intruder can say infinitely many different things at any point.

In this paper, we show how to combine and extend different methods to build a highly effective security protocol model-checker. Our starting point is the approach of [4] of using *lazy data-types* to model the infinite state-space associated with a protocol. A lazy data-type is one where data-type constructors (e.g. *cons* for building lists, or *node* for building trees) build data-types without evaluating their arguments; this allows one to represent and compute with infinite data (e.g. streams or infinite trees), generating arbitrary prefixes of the data on demand. In [4], lazy data-types are used to build, and compute with, models of security protocols: a protocol and description of the powers of an intruder are formalized as an infinite tree. Lazy evaluation is used to decouple the model from search and heuristics, building the infinite tree on-the-fly, in a demand-driven fashion.

* This work was supported by the FET Open Project IST-2001-39252, "AVISPA: Automated Validation of Internet Security Protocols and Applications" [2].

This approach is conceptually and practically attractive as it cleanly separates model construction, search, and search reduction techniques. Unfortunately, it doesn't address the problem of the prolific Dolev-Yao intruder and hence 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 technique*, uses a symbolic representation to avoid explicitly enumerating the possible messages the intruder can generate, by representing intruder messages using terms with variables, and storing and manipulating constraints about what must be generated and from which knowledge.

The lazy intruder is a general, technology-independent technique that can be effectively incorporated in different approaches to protocol analysis. Here, we show how to 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. [17, 7, 1, 22, 16, 8, 10] to the symbolic representation of the intruder so that our lazy intruder technique is applicable to a larger class of protocols and properties. Second, despite the extensions, we 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, which provides an infinite-state transition system. On the practical side, we also investigate the question of an efficient implementation of the lazy intruder, i.e. how to organize state exploration and constraint reduction.

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 (but one) known attacks and discovers a new one (on the Yahalom protocol) in a test-suite of 36 protocols from the Clark/Jacob library [9] in under one minute of CPU time for the entire suite. Moreover, we have successfully applied OFMC to large-scale protocols including 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 §5 our analysis of the H.530 protocol [18], a protocol invented by Siemens and proposed as an Internet standard for multimedia communications. We have modeled the protocol in its full complexity and have detected a replay attack in 1.6 seconds. The weakness is serious enough that Siemens has changed the protocol.

We proceed as follows. In §2 we give the formal model that we use for protocol analysis. In §3 we review the lazy protocol analysis approach. In §4 we formalize the lazy intruder and how constraints are reduced. We present experimental results in §5, and discuss related and future work in §6. Due to lack of space, examples and proofs have been shortened or omitted; details can be found in [5].

2 Protocol Specification Languages and Model

The formal model we use for protocol analysis with our tool OFMC is based on two specification languages, which we have been developing in the context of the AVISPA project [2]: a high-level protocol specification language (HLPSL) and a low-level one (the Intermediate Format IF). HLPSL allows the user to specify the protocols in an Alice & Bob style notation. A translator called HLPSL2IF automatically translates HLPSL specifications into the IF, which the OFMC tool takes as input. Due to space limitations and since the ideas behind our protocol specification languages are fairly standard, e.g. [11, 19], we restrict ourselves here to the presentation of the IF; discussions and examples can be found in the technical report [5].

The Syntax of the IF 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:

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

We write $\mathrm{vars}(t)$ to denote the set of variables occurring in a (message, fact, or state) *term* t, and say that t is *ground* when $\mathrm{vars}(t) = \emptyset$.

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

Messages in the IF are either atomic messages or are composed using *pairing* $\langle M_1,M_2\rangle$, or the *cryptographic operators* $\{\!|M_1|\!\}_{M_2}$ and $\{M_1\}_{M_2}$ (for *symmetric* and *asymmetric encryption* of M_1 with M_2), or M^{-1} (the *asymmetric inverse* of M). 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, which leads to smaller search spaces at the cost of abstracting away any type-flaw attacks on the protocol.

Note also that we follow the standard *perfect cryptography assumption*, i.e. the only way to decrypt an encrypted message is to have the appropriate key. Moreover, like most other approaches, we here 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. Note too that unlike other models, e.g. [12, 22], we are not bound to a fixed public-key infrastructure where every agent initially has a key-pair and knows the public key of every other agent. Rather, we can also consider protocols where keys are generated, distributed, and revoked.

The IF contains both positive and negative *facts*. A *positive fact* represents either the local state of an honest agent, a message on the network (i.e. one sent but not yet received), or that a message is known by the intruder. To ease the formalization of protocols, we introduce *negative facts* as well as additional fact symbols, expressing, e.g. secrecy or set membership; see [5] for details. To illustrate how these additions allow for the explicit encoding of problems in a natural way, consider the Needham-Schroeder public-key protocol with a key-server [9]. In a realistic model of this protocol, an agent should maintain a database of known public keys, which is shared over all protocol executions he participates in, and 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 $\mathsf{knows\_pk}$.

Note also that the set of composed messages can similarly be easily extended, e.g. with cryptographic primitives for hashes and key-tables, without affecting the theoretical results we present below. In this paper, we focus on this smaller language for brevity.

A *state* is a set of positive facts, which we denote as a sequence of positive facts, separated by dots. Note that in our approach we actually employ set rewriting instead of multiset rewriting, as the HLPSL2IF translator ensures that in no reachable state the same positive fact can appear more than once, so we need not distinguish between multisets and sets.

We define in the usual way the notions related to substitution and unification, such as *ground term*, *ground substitution*, *unifier*, *most general unifier (mgu)*, and *matching*; see, e.g., [3]. We denote the application of a substitution σ to a term t by writing $t\sigma$ and denote the composition of substitutions σ_1 and σ_2 by writing $\sigma_1\sigma_2$. As we only consider substitutions with finite domains, we represent a substitution σ with $\mathrm{dom}(\sigma) = \{v_1, \dots, v_n\}$ by $[v_1 \mapsto v_1\sigma, \dots, v_n \mapsto v_n\sigma]$. The *identity substitution* id is the substitution with $\mathrm{dom}(\mathrm{id}) = \emptyset$. We say that two substitutions σ_1 and σ_2 are *compatible*, written $\sigma_1 \approx \sigma_2$, if $v\sigma_1 = v\sigma_2$ for every $v \in \mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2)$. 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.

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

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

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

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

where N_1 is a set of negative facts, and P_1 and P_2 are sets of positive facts that do not contain state or msg facts. Moreover, if $\mathsf{i\_knows}(m) \in P_1$ then $\mathsf{i\_knows}(m) \in P_2$, which ensures that the intruder knowledge is *monotonic*, i.e. that the intruder never forgets messages during a transition.

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

Rules of the form (1) are adequate to describe large classes of protocols (including those discussed in §5); see [5] for examples of actual IF rules.

The Dolev-Yao Intruder We follow Dolev and Yao [12] and consider the standard, protocol-independent, asynchronous model in which the intruder controls the network but cannot break cryptography. In particular, he can intercept messages and analyze them if he possesses the corresponding decryption keys, and he can generate messages and send them under any agent name.

Definition 1. *For a set M of messages, let $\mathcal{DY}(M)$ (for Dolev-Yao) be the smallest set closed under the following* generation (G) *and* analysis (A) rules*:*

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

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

The generation rules express that the intruder can compose messages from known messages using pairing and symmetric encryption; the analysis rules describe how he can decompose messages. For brevity, we have omitted the rules for asymmetric encryption and decryption, which are straightforward. Note that this formalization correctly handles non-atomic keys, for instance $m \in \mathcal{DY}(\{\{\!|m|\!\}_{(\langle k_1, k_2 \rangle)}, k_1, k_2\})$, as opposed to other models such as [1, 20, 24, 26] that only handle atomic keys.

The Semantics of the IF Using $\mathcal{DY}$, we now define the protocol model provided by the IF in terms of an infinite-state transition system, where the IF rules define a state-transition function. In this definition, we incorporate an optimization that we call *step-compression*, which is based on the idea [1, 7, 8, 10, 22] that we can identify the intruder and the network: every message sent by an honest agent is received by the intruder and every message received by an honest agent comes from the intruder. Formally, we compose (or "compress") several steps: when the intruder sends a message, an agent reacts to it according to his rules, and the intruder diverts the agent's answer. A bisimulation proof shows that, for large classes of properties, the model with such composed transitions (which we present here) is "attack-equivalent" to the model with single (uncompressed) transitions, i.e. we end up in an attack state using composed transitions iff that was the case using uncomposed transitions.

Definition 2. *The* successor function $succ_R(S) = \bigcup_{r \in R} step_r(S)$ *maps a set of rules R and a state S to a set of states, where*

$$\begin{aligned} step_r(S) = \{S' \mid \exists \sigma.\, \mathrm{ground}(\sigma) \;\wedge\; \mathrm{dom}(\sigma) = \mathrm{vars}(m_1) \cup \mathrm{vars}(m_2) \cup \mathrm{vars}(P_1) & \quad (2)\\ \wedge\;\; m_1\sigma \in \mathcal{DY}(\{\, i \mid \mathsf{i\_knows}(i) \in S\}) & \quad (3)\\ \wedge\;\; \mathsf{state}(m_2\sigma) \in S \;\wedge\; P_1\sigma \subseteq S \;\wedge\; \forall f.\ \mathsf{not}(f) \in N_1 \implies f\sigma \notin S & \quad (4)\\ \wedge\;\; S' = (S \setminus (\mathsf{state}(m_2\sigma) \cup P_1\sigma)) \cup \mathsf{state}(m_3\sigma) \cup \mathsf{i\_knows}(m_4\sigma) \cup P_2\sigma\} & \quad (5) \end{aligned}$$

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

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

We define the *set of reachable states* associated with a protocol (I, R, G) as $reach(I, R) = \bigcup_{n \in \mathbb{N}} succ_R^n(I)$. The set of reachable states is ground as no state reachable from the initial state I may contain variables (by the conditions (i) and (ii) in the definition of protocol). As the properties we are interested in are reachability properties, we will sometimes abstract away the details of the transition system and refer to this set as the *ground model* of the protocol.

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

3 The Lazy Infinite-State Approach

The transition system defines a (computation) tree in the standard way, where the root is the initial system state and children represent the ways that a state can evolve in one transition. The tree has infinitely many states since, by the definition of $\mathcal{DY}$, every node has infinitely many children. It is also of infinite depth, provided we do not 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* [4] allows us to handle the infinitely long branches. As we have integrated the lazy intruder with our previous work, we now briefly summarize the main ideas of [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 a goal state. Our on-the-fly model-checker OFMC uses iterative deepening to search this infinite tree for an attack state. When an attack is found, OFMC returns the attack trace, i.e. the sequence of exchanged messages leading to the attack state. This yields a semi-decision procedure for protocol insecurity: our procedure always terminates (at least in principle) when an attack exists. Moreover, our search procedure terminates for finitely many sessions (formally: if there are finitely many agents and none of them can perform an unbounded number of transitions) when we employ the lazy intruder to restrict 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.

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, hence the intruder is called *lazy*.

The idea behind the lazy intruder was, to our knowledge, first proposed by [17] and then subsequently developed by [7, 1, 22, 16, 8, 10]. Our contributions to the symbolic intruder technique are as follows. First, we simplify the technique, which, as we formally show in [5], 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 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 the introduction of negative facts alongside standard positive facts in the IF rewrite rules leads to inequality constraints in the lazy intruder.

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

Definition 3. *The* semantics of a constraint $\mathit{from}(T, \mathit{IK})$ *is the set of satisfying ground substitutions* σ *for the variables in the constraint, i.e.* $[\![\mathit{from}(T, \mathit{IK})]\!] = \{\sigma \mid \mathrm{ground}(\sigma) \wedge \mathrm{ground}(T\sigma \cup \mathit{IK}\sigma) \wedge (T\sigma \subseteq \mathcal{DY}(\mathit{IK}\sigma))\}$*. A* constraint set *is a finite set of constraints and its semantics is the intersection of the semantics of its elements, i.e., overloading notation,* $[\![\{c_1, \ldots, c_n\}]\!] = \sqcap_{i=1}^{n} [\![c_i]\!]$*. A constraint set* C *is* satisfiable *if* $[\![C]\!] \neq \emptyset$*. A constraint* $\mathit{from}(T, \mathit{IK})$ *is* simple *if* $T \subseteq \mathcal{V}$*, and we then write* $\mathrm{simple}(\mathit{from}(T, \mathit{IK}))$*. A constraint set* C *is simple if all its constraints are simple, and we then write* $\mathrm{simple}(C)$*.*

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 2, every simple constraint set is satisfiable.) This reduction is performed using the generation and analysis rules of Fig. 1, which describe possible transformations of the constraint set (for brevity, we have again omitted the rules for asymmetric encryption and decryption, which are straightforward). Afterwards, we show that this reduction does not change the set of solutions, roughly speaking $[\![C]\!] = [\![\mathit{Red}(C)]\!]$, for a relevant class of constraints C.

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

The generation rules G^l_{pair} and G^l_{scrypt} express that the constraint stating that the intruder can generate a message composed from submessages m_1 and m_2 (using pairing and symmetric encryption, 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 if this

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

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

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

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

Fig. 1. Lazy intruder: constraint reduction rules

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: the intruder's choice for this variable does not matter at this stage of the search and hence we postpone this decision.

The analysis of the intruder knowledge is more complex for the lazy intruder than in the ground model, as messages may now contain variables. In particular, if the key of an encrypted message is a variable, then whether or not the intruder can decrypt this message is determined by the substitution we (later) choose for this variable. We solve this problem by using the rule A^l_{scrypt}, where the variable key can be instantiated during further constraint reduction.[1] More specifically, for a message $\{\!|m|\!\}_k$ that the intruder attempts to decrypt, we add the content m to the intruder knowledge of the respective constraint (as if the check was already successful) and add a new constraint expressing that the symmetric key k necessary for decryption must be generated from the same knowledge. Hence, if we attempt to decrypt a message that cannot be decrypted using the corresponding intruder knowledge, we obtain an unsatisfiable constraint set.

Definition 4. *Let $\vdash$ denote the derivation relation described by the rules in Fig. 1. The set of pairs of simple constraint sets and substitutions derivable from (C, id) is $\mathit{Red}(C) = \{(C', \sigma) \mid ((C, \mathrm{id}) \vdash (C', \sigma)) \wedge \mathrm{simple}(C')\}$.*

Properties of *Red* By Theorem 1 below, proved in [5], 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 3).

[1] 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.

Definition 5. *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: (i) $IK_i \subseteq IK_j$ for $i \leq j$, and (ii) $\mathrm{vars}(IK_i) \subseteq \cup_{j=1}^{i-1} \mathrm{vars}(T_j)$.*

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

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

The Lazy Intruder Reachability We describe now the integration of constraint reduction into the search procedure for reachable states. The space of *lazy states* consists of states that may contain variable symbols (as opposed to the ground model where all reachable states are ground) and that are associated with a set of *from* constraints as well as a collection of inequalities. The inequalities will be used to handle negative facts in the context of the lazy intruder. We assume that the inequalities are given as a conjunction of disjunctions of inequalities between terms. We will use the inequations to rule out certain unifications, e.g. to express that both the substitutions $\sigma = [v_1 \mapsto t_1, v_2 \mapsto t_2]$ and $\tau = [v_1 \mapsto t_3]$ are excluded in a certain state, we use the inequality constraint $(v_1 \neq t_1 \boldsymbol{\vee} v_2 \neq t_2) \boldsymbol{\wedge} (v_1 \neq t_3)$, where we write $\boldsymbol{\vee}$ and $\boldsymbol{\wedge}$ 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 6. *A* lazy state *is a triple (P, C, N), where P is a sequence of (not necessarily ground) positive facts, C is a constraint set, and N is a conjunction of disjunctions of inequalities between terms. The semantics of a lazy state is $[\![(P, C, N)]\!] = \{P\sigma \mid \sigma \in [\![C]\!] \wedge \sigma \models N\}$, where $\sigma \models N$ is defined for a substitution σ as expected.*

Let $\mathrm{freshvars}_r(S)$ *be a renaming of the variables in a lazy state $S = (P, C, N)$ with respect to a rule r such that* $\mathrm{vars}(\mathrm{freshvars}_r(S))$ *and* $\mathrm{vars}(r)$ *are disjoint. The* lazy successor function $\mathit{lsucc}_R(S) = \cup_{r \in R}\, \mathit{lstep}_r(\mathrm{freshvars}_r(S))$ *maps a set of rules R and a lazy state $S = (P, C, N)$ to a set of lazy states, where*

$$
\begin{aligned}
\mathit{lstep}_r(P, C, N) = \{(P', C', N') \mid \exists\, \sigma.\\
\mathrm{dom}(\sigma) \subseteq \mathrm{vars}(m_1) \cup \mathrm{vars}(m_2) \cup \mathrm{vars}(P_1) \cup \mathrm{vars}(P) \cup \mathrm{vars}(C) \cup \mathrm{vars}(N)\\
\wedge\ C' = (C \cup \mathit{from}(m_1, \{i \mid \mathsf{i\_knows}(i) \in P\}))\sigma && (6)\\
\wedge\ \mathsf{state}(m_2\sigma) \in P\sigma\ \ \wedge P_1\sigma \subseteq P\sigma && (7)\\
\wedge\ N' = N \boldsymbol{\wedge} \textstyle\bigwedge_{\phi \in \mathit{subCont}(N_1\sigma, P\sigma)} \phi && (8)
\end{aligned}
$$

$$\wedge\ P' = (P\sigma \setminus (\mathsf{state}(m_2\sigma) \cup P_1\sigma)) \\ \cup\ \mathsf{state}(m_3\sigma) \cup \mathsf{i\_knows}(m_4\sigma) \cup P_2\sigma\} \tag{9}$$

for a rule r of the form $\mathsf{msg}(m_1).\mathsf{state}(m_2).P_1.N_1 \Rightarrow \mathsf{state}(m_3).\mathsf{msg}(m_4).P_2$, where $subCont(N,P) = \{\phi \mid \exists\, t, t', \sigma.\ \mathsf{not}(t) \in N\ \wedge\ t' \in P\ \wedge\ t\sigma = t'\sigma$

$$\wedge\ \exists\, v_1, \ldots, v_n, t_1, \ldots, t_n.\ \sigma = [v_1 \mapsto t_1, \ldots, v_n \mapsto t_n]\ \wedge\ \phi = \bigvee_{i=1}^{n} v_i \neq t_i\}$$

Similar to the successor function of the ground model, the lazy successor function also performs step-compression, i.e. it performs three operations in one transition: the intruder sends a message, an honest agent reacts to it, and the intruder adds the answer to his knowledge. The most notable change is the renaming of the variables of the rules to avoid clashes with the variables that may appear in the lazy states. More in detail, the constraint in condition (6) expresses that the message m_1 that occurs on the LHS of the rule r must be generated by the intruder from his current knowledge. Condition (7) is similar to the first two conjuncts in condition (4) in the ground model, where the substitution is now applied also to the set of positive facts in the state (i.e., instead of matching, we now perform unification). Condition (8) states that the inequalities are conjoined with the conjunction of all formulae that $subCont(N_1\sigma, P\sigma)$ yields. For a set of negative facts N and a set of positive facts P, $subCont(N,P)$ generates a disjunction of inequalities that excludes all unifiers between two positive facts t and t' such that $\mathsf{not}(t) \in N$ and $t' \in P$. Note that in the special case that $t = t'$ we obtain the solution $\sigma = []$, and naturally we define $\bigvee_{i=1}^{0} \phi$ to be simply false for any ϕ. Finally, condition (9) describes the positive facts P' in the successor state, which result by removing the positive LHS facts from P (under σ) and adding the RHS facts (under σ).

We define the set of *reachable lazy states* associated to a protocol (I, R, G) as $lreach(I,R) = \bigcup_{n \in \mathbb{N}} lsucc_R^n(I, \emptyset, \emptyset)$. We also call $lreach(I,R)$ the *lazy model* of the protocol (I, R, G). The lazy model is equivalent to the ground model, i.e. they represent the same set of reachable states.

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

Recall that we have defined that a protocol is secure iff *goalcheck* (which represents the negation of the property the protocol aims to provide, i.e. it represents the attacks on the protocol) is empty for all reachable ground states. A similar check suffices in the lazy intruder model. We define the lazy goal-check for a lazy state $S = (P, C, N)$ and a goal state g as $lgoalcheck_g(P,C,N) = \{\sigma \mid g\sigma \subseteq P\sigma\}$. If *lgoalcheck* is not empty in a reachable lazy state S, then either S represents an attack or S is *unsatisfiable*, i.e. its semantics is the empty set.

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

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

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

Second, we must represent the reachable states. The lazy infinite-state approach provides a straightforward solution to this problem, where 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.) We can apply the lazy goal-check as a filter on this tree to obtain the lazy goal states.

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

Lemma 2. *Let (P, C, N) be a lazy state where C is simple and N is satisfiable (i.e. $\exists \sigma.\ \sigma \models N$). Then $[\![(P, C, N)]\!] \neq \emptyset$.*

The proof, given in [5], 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 key idea behind inequalities in 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 $(C', \tau) \in Red(C)$ and $N\tau$ is satisfiable, then $[\![(P, N, C)]\!] \neq \emptyset$, since C' is simple and $[\![C']\!] \subseteq [\![C]\!]$, by Theorem 1. Otherwise, if $Red(C) = \emptyset$ or if N is unsatisfiable, then $[\![(P, C, N)]\!] = \emptyset$, also by Theorem 1.

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

Lemma 3. *For a protocol (I, R, G), if $(P, C, N) \in lreach_R(I)$ then C is well-formed.*

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

Organizing State Exploration and Constraint Reduction When implementing the lazy intruder we are faced with two design decisions: (i) in which order the two "filters" mentioned above are applied, and (ii) how constraint reduction should be realized.

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. In OFMC we apply *Red* after each transition to check if the constraints are still satisfiable. This allows us to eliminate from the search all states with unsatisfiable constraint sets, as the successors of such states 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 goal states (using *lgoalcheck*). 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 of the search tree, we continue the search with the original constraint set.

With respect to (ii), note that the question of how to compute the constraint reduction (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 as long as they are applicable. The result is that the intruder knowledge is "normalized" with respect to the analysis rules. As a consequence, we need not 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 m' appears that can be unified with m. In contrast, with analysis on demand it is in this case necessary to check if a unifiable message may be obtained through analysis.

However, when normalizing the intruder knowledge, we must take into account that the analysis may lead to substitutions. Every substitution restricts the set of possible solutions and in this case the restriction is only necessary if the respective decrypted content of the message is actually used later (which is, in contrast, elegantly handled by applying the analysis on demand).[2] Our solution to this problem is to distinguish between analysis steps that require a substitu-

[2] 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.

tion and those that do not. The latter can be performed without restriction, the former are not performed; rather, we add to the intruder knowledge the term that would be obtained in case of a successful analysis (without unification), and *mark* the term to express that the intruder *may* know it, but only under a certain substitution. If a marked term is actually needed, then the respective analysis steps are performed, else the marked term stays in the intruder knowledge.

Our strategy, which is a combination of demand-driven analysis and normalization, is somewhat complex but very efficient, as the occurrence of marked terms is rare and the use of a marked term is even rarer.

5 Experimental Results

To assess the effectiveness and performance of OFMC, we have tested it on a large protocol suite, which includes the protocols of the Clark/Jacob library [9, 13], as well as a number of industrial-scale protocols. Since OFMC implements a semi-decision procedure, it does not terminate for correct protocols, although it can establish the correctness of protocols for a bounded number of sessions. We give below search times for finding attacks on flawed protocols.

The Clark/Jacob Library The OFMC can find an attack for 32 of the 33 flawed protocols of the Clark/Jacob library.[3] As the times in Table 5 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. (Times are obtained on a PC with a 1.4GHz Pentium III processor and 512Mb of RAM, but note that, due to the use of iterative deepening search, OFMC requires a negligible amount of memory.) To our knowledge, no other tool for finding attacks is this fast and has comparable coverage.

Note that the analysis of the untyped and typed IF specifications may lead to the detection of different kinds of attacks. When this is the case, in Table 5 we report the two attacks found. (In all other cases, the times are obtained using the default untyped model.) Also note that the table contains four variants of protocols in the library, marked with a "*", that we have additionally analyzed, and that "MITM" abbreviates man-in-the-middle attack and "STS" abbreviates replay attack based on a short-term secret. Table 5 also reports a new attack that we have found on the Yahalom protocol, which we describe in [5].

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 [21]), and in particular, the H.530 protocol of the ITU [18]. H.530, which has been developed by Siemens, provides mutual authentication and key agreement in mobile roaming scenarios in multimedia communication.

[3] Missing is the CCITT X.509 protocol, where agents may sign messages they cannot analyze completely. OFMC cannot find this attack since the HLPSL does not (yet) allow us to specify an appropriate goal that is violated by this weakness.

Protocol Name	Attack	Time
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

Protocol Name	Attack	Time
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
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 Clark/Jacob library

H.530 is deployed as shown in the left part of Fig. 2: 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. 2 shows the messages exchanged: first, both MT and VGK create Diffie-Hellman half-keys, along with hashes that are encrypted for the AuF (denoted by the messages Req_{MT} and Req_{VGK}, respectively). After a successful check of these messages, AuF replies with appropriate acknowledge messages Ack_{MT} and Ack_{VGK} that also contain encrypted hashes for the respective 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).

We have applied OFMC to automatically analyze the H.530 protocol in collaboration with Siemens. The main problem that we had to tackle for this analysis is the fact that the protocol employs the Diffie-Hellman key-agreement, which is based on a property of cryptographic algorithms (namely the commutativity of exponents) that violates the free algebra assumption. We lack space here to discuss in detail how we solved this problem in our model. The central point is this: while the messages exchanged in the H.530 protocol are considerably more complex than the ones of the Clark/Jacob protocols, this complexity is not a problem for our approach, unlike for other model-checking tools, e.g. [13, 20]. We could directly analyze the original specification of the H.530 without first simplifying the messages.

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

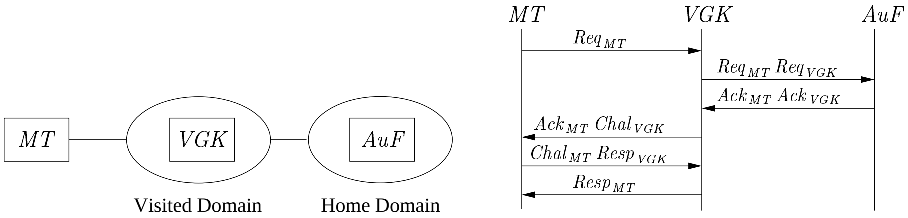

Fig. 2. The H.530 protocol (simplified). The deployment of the protocol is on the left, the messages exchange between the participants are summarized are on the right.

intruder impersonating *mt* can negotiate a new Diffie-Hellman key with *vgk*, "hijacking" *mt*'s identity. To perform the attack, the intruder must at least be able to eavesdrop and insert messages both on the connection between MT and VGK, and on the connection between VGK and AuF. We have suggested including MT's Diffie-Hellman half-key in the encrypted hash of the message Ack_{VGK} to fix this problem. With this extension we have not found any further weaknesses of the protocol and Siemens has changed the protocol accordingly.

6 Related Work and Concluding Remarks

There are several model-checking approaches similar to ours. As a prominent example, we compare our approach with Casper [13, 20], a compiler that maps protocol specifications, written in a high-level language similar to HLPSL (which was inspired by CAPSL [11]), into descriptions in the process algebra CSP. The approach uses finite-state model-checking with FDR2. Casper/FDR2 has successfully discovered flaws in a wide range of protocols: among the protocols of the Clark/Jacob library, it has found attacks on 20 protocols previously known to be insecure, as well as attacks on 10 other protocols originally reported as secure. Experiments indicate that OFMC is considerably faster than Casper/FDR2, despite being based on a more general model: Casper limits the size of messages to obtain a finite-state model. This limitation is problematic for the detection of type-flaw attacks, e.g. Casper/FDR also misses our type-flaw attack on Yahalom (cf. [5]). Finally, Casper does not support non-atomic keys, which hinders its application to protocols like IKE, where each participant constructs only a part of the shared key that is negotiated.

The Athena tool [26] combines model-checking and interactive theorem-proving techniques with strand spaces [15] to reduce the search space and automatically prove the correctness 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 expand on the remarks in §4. The idea of a symbolic intruder model has undergone a steady evolution, becoming increasingly simpler and general. In the earliest work [17], both the technique itself and the proof were of substantial complexity. In [1, 7], both based on process calculi, the technique and its formal presentation were considerably simplified. [22] generalized the approach to support non-atomic symmetric keys, while [10] improved the approach of [22] by increasing its expressiveness and providing a more efficient implementation. [8] lifted the restriction of a fixed public-key infrastructure, where every agent has a fixed key-pair and knows his own private key and each agent's public key. This work is the closest to ours. Both approaches are based on HLPSL/IF and (multi)set rewriting. However, there are important differences. For instance, we have removed all procedural aspects from the symbolic intruder rules, making the approach more declarative and the proofs simpler. Moreover, we have extended the lazy intruder by introducing inequalities, which, together with the notion of simple constraints, has a very natural interpretation: "the intruder can generate as many different terms as he likes".

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

To summarize, we have presented an approach to security protocol analysis implemented by the model-checker OFMC, which represents a substantial development of the idea of on-the-fly model-checking proposed in [4]. 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 suggests. Current work involves applying OFMC to other industrial-scale protocols, such as those proposed by the IETF. Although initial experience is positive, we see an eventual role for heuristics in leading to further improvements. For example, a simple evaluation function could be: "has the intruder learned anything new through this step, and how interesting is what he learned?" We have also been investigating the integration of partial-order reduction techniques in our model-checker and the first results are very positive [6].

References

1. R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. Concur'00*, LNCS 1877, pp. 380–394. Springer, 2000.
2. AVISPA: Automated Validation of Internet Security Protocols and Applications. FET Open Project IST-2001-39252, `www.avispa-project.org`.
3. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge U. Pr., 1998.

4. D. Basin. Lazy infinite-state analysis of security protocols. In *Proc. CQRE'99*, LNCS 1740, pp. 30–42. Springer, 1999.
5. D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis (Extended Version). Technical Report 404, ETH Zurich, Computer Science, 2003. `http://www.inf.ethz.ch/research/publications/`.
6. D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. Technical Report 405, ETH Zurich, Computer Science, 2003. `http://www.inf.ethz.ch/research/publications/`.
7. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proc. ICALP'01*, LNCS 2076, pp. 667–681. Springer, 2001.
8. Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proc. ASE'01*. IEEE Computer Society Press, 2001.
9. 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`.
10. R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proc. SAS 2002*, LNCS 2477, pp. 326–341. Springer, 2002.
11. G. Denker, J. Millen, and H. Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, 2000.
12. D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
13. B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR. In *Proc. FMSP'99 (Formal Methods and Security Protocols)*.
14. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Proc. FMSP'99 (Formal Methods and Security Protocols)*.
15. 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.
16. M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proc. CSFW'01*. IEEE Computer Society Press, 2001.
17. A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.
18. ITU-T Recommendation H.530: Symmetric Security Procedures for H.510 (Mobility for H.323 Multimedia Systems and Services). 2002.
19. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proc. LPAR 2000*, LNCS 1955, pp. 131–160. Springer, 2000.
20. G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.
21. C. Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In *Proc. 1999 IEEE Symposium on Security and Privacy*.
22. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. CCS'01*, pp. 166–175, ACM Press, 2001.
23. J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. In *Proc. 1997 IEEE Symposium on Security and Privacy*.
24. L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.
25. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. Modelling and Analysis of Security Protocols. Addison Wesley, 2000.
26. 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.