Integrating Automated and Interactive Protocol Verification (Extended Version)

Achim D. Brucker[1] and Sebastian A. Mödersheim[2]

[1] SAP Research, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany
achim.brucker@sap.com
[2] IBM Research, Säumerstrasse 4, 8803 Rüschlikon, Switzerland
smo@zurich.ibm.com

Abstract. A number of current automated protocol verification tools are based on abstract interpretation techniques and other over-approximations of the set of reachable states or traces. The protocol models that these tools employ are shaped by the needs of automated verification and require subtle assumptions. Also, a complex verification tool may suffer from implementation bugs so that in the worst case the tool could accept some incorrect protocols as being correct. These risks of errors are also present, but considerably smaller, when using an LCF-style theorem prover like Isabelle. The interactive security proof, however, requires a lot of expertise and time.
We combine the advantages of both worlds by using the representation of the over-approximated search space computed by the automated tools as a "proof idea" in Isabelle. Thus, we devise proof tactics for Isabelle that generate the correctness proof of the protocol from the output of the automated tools. In the worst case, these tactics fail to construct a proof, namely when the representation of the search space is for some reason incorrect. However, when they succeed, the correctness only relies on the basic model and the Isabelle core.

1 Introduction

Over the last decade, a number of automated tools for security protocol verification have been developed such as AVISPA [1] and ProVerif [4]. They allow engineers to find problems in their security protocols before deployment. Indeed, several attacks to security protocols have been detected using automated tools. The focus of this work is the positive case—when no attack is found: to obtain a proof of security.

Many automated tools employ over-approximation and abstraction techniques to cope with the infinite search spaces that are caused, e.g., by an unbounded number of protocol sessions. This means to check the protocol in a finite abstract model that (in a sense) subsumes the original model. Thus, if the protocol is correct in the abstract model, then so it is in the original model. The soundness of such abstractions depends on subtle assumptions, and it is often hard to keep track of them, even for experts. Moreover, the abstract protocol models are tuned to the method and hard to use correctly. Finally, tools may also have bugs. For all these reasons, it is not unlikely that insecure protocols are accidentally verified by automated verification tools.

There are semi-automated methods such as the Isabelle theorem prover which offer a high reliability: if we trust in a small core (the proof checking and some basic logical axioms), we can rely on the correctness of proved statements. However, conducting proofs in Isabelle requires considerable experience in both formal logic and proof tactics, as well as a proof idea for the statement to show.

In this work, we combine the best of both worlds: reliability and full automation. The idea is that abstraction-based tools—supposedly—compute a finite representation of an over-approximated search space, i.e., of what can happen in a given protocol, and that this representation can be used as the basis to automatically generate a security proof in Isabelle. This proof is w.r.t. a "clean" standard protocol model without over-approximation. If anything goes wrong, e.g., the abstraction is not sound, this proof generation fails. However, if we succeed in generating the proof, we only need to trust in the standard protocol model and the Isabelle core. Our vision is that such automatically generated verifiable proofs can be the basis for reaching the highest assurance level EAL7 of a common criteria certification at a low cost, i.e., easy-to-use automated tools.

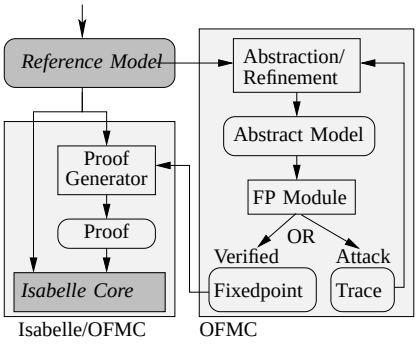

Fig. 1. The workflow of a protocol verification approach combing automated (e.g., OFMC) and interactive (e.g., Isabelle) techniques.

We have realized this integration of automated and interactive protocol verification in a prototype tool that is summarized in Fig. 1. The protocol and the proof goals (e.g., secrecy, authentication) are specified in the *reference model* (Sect. 2 and Sect. 3). This reference model is not driven by technical needs of the automated verification and is close to other high-level protocol models. The description is fed into the automated tool, where we consider (a novel module of) the Open-source Fixed-point Model-Checker OFMC [21], formerly called On-the-Fly Model-Checker. OFMC first chooses an initial abstraction and produces an abstracted version of the protocol description as a set of Horn clauses. From this, the fixed-point (FP) module computes a least fixed-point of derivable events. If this fixed-point contains an attack, this can either be a real attack (that similarly works in the reference model) or a false attack that was caused by the abstraction. By default, OFMC will assume that the attack is false, refine the abstraction based on the attack details and restart the verification. If the computed fixed-point does not contain an attack, it is handed to the proof generator of the Isabelle/OFMC, our extension of the interactive theorem prover Isabelle [22]. The proof generator translates the fixed-point into the terms of the reference model (using annotations about the abstraction OFMC considered) and generates an Isabelle proof with respect to the protocol and goal description in the reference model. This proof is fed into the Isabelle core to check the proof.

We emphasize two points. First, the entire approach is completely automatic: after the specification of the protocol and goals in the reference model, no further user interaction is required. Second, we need to trust in only two points—marked with a dark-gray background in Fig. 1: the reference model and the Isabelle core. Bugs in any other part, namely OFMC or the proof generation, can in the worst case result in a failure to verify a (possibly correct) protocol, but they cannot make us falsely accept a flawed protocol as being correct.

Our approach currently has two limitations: first, like in the similar Isabelle formalizations of Paulson and Bella, we consider a typed model (excluding type-flaw attacks), and second, based on this, we limited the intruder composition to well-typed messages of the protocol. The first limitation can be justified by implementation discipline (see [17]). The second limitation is without loss of generality as we show in Theorem 1; also current experiments suggest that we can eliminate this limitation in a future version.

Contributions. An increasing number of works considers the combination of automated methods with interactive theorem proving to obtain both highly reliable and fully automated verification. In this paper, we contribute to this line of work with a novel approach for security protocols. The first novel aspect is that our approach automatically generates a proof from the representation of an over-approximation of the search space computed by an automated protocol verifier. Techniques based on over-approximation, similar to the ones we consider, have turned out to be very successful in protocol verification [4,6,7,9], and our approach is thus the first step towards employing a whole class of established tools for automated proof generation. The second novel aspect is that the proof is entirely based on a standard protocol model without over-approximation close to the model employed for instance in [23]. Our approach thus relates over-approximated representations with standard protocol models.

Practically, we have implemented the integration between Isabelle on the interactive side and the novel FP-module of OFMC on the automated side. The result is a completely automated protocol verifier for an unbounded number of sessions and agents that produces Isabelle-verifiable proofs with respect to a standard protocol model.

2 The Reference Protocol Model

We begin with a reference protocol model, which is used in the Isabelle theorem prover and is thus the basis of this work. The model is inspired by the formalization of several security protocols in Isabelle by Paulson and others in [23,3] and is close to the persistent IF model in [20].

2.1 Messages

We follow the common black-box cryptography model where messages are modeled as symbolic terms. We define the set of all messages in style of an inductive datatype in a functional programming language. We discuss the subtle difference with respect to other definitions below.

Definition 1. *Let $\mathcal{F}$, $\mathcal{L}_\mathcal{A}$, $\mathcal{L}_\mathcal{N}$, $\mathcal{L}_\mathcal{S}$, $\mathcal{L}_\mathcal{P}$, $\mathcal{V}_\mathcal{A}$, $\mathcal{V}_\mathcal{N}$, $\mathcal{V}_\mathcal{S}$, $\mathcal{V}_\mathcal{P}$, and $\mathcal{V}_\mathcal{U}$ be pairwise disjoint sets of symbols where $\mathcal{F}$, $\mathcal{L}_\mathcal{A}$, $\mathcal{L}_\mathcal{N}$, $\mathcal{L}_\mathcal{S}$, and $\mathcal{L}_\mathcal{P}$ are finite, and $\mathcal{V}_\mathcal{A}$, $\mathcal{V}_\mathcal{N}$, $\mathcal{V}_\mathcal{S}$, $\mathcal{V}_\mathcal{P}$, and $\mathcal{V}_\mathcal{U}$ are countable. We define the sets of* messages, agents, nonces, symmetric keys, *and* public keys, *respectively, to be the following sets of terms:*

$$\begin{aligned}\mathcal{M} &= \mathsf{agent}\ \mathcal{A} \mid \mathsf{nonce}\ \mathcal{N} \mid \mathsf{symkey}\ \mathcal{S} \mid \mathsf{pubkey}\ \mathcal{P} \mid \mathcal{V}_\mathcal{U} \\ &\quad \mid \mathsf{crypt}\ \mathcal{M}\ \mathcal{M} \mid \mathsf{inv}\ \mathcal{M} \mid \mathsf{scrypt}\ \mathcal{M}\ \mathcal{M} \mid \mathsf{cat}\ [\mathcal{M}] \mid \mathcal{F}\ \mathcal{M} \\ \mathcal{A} &= \mathcal{L}_\mathcal{A} \times \mathbb{N} \mid \mathcal{V}_\mathcal{A} \quad \mathcal{N} = \mathcal{L}_\mathcal{N} \times \mathbb{N} \mid \mathcal{V}_\mathcal{N} \quad \mathcal{S} = \mathcal{L}_\mathcal{S} \times \mathbb{N} \mid \mathcal{V}_\mathcal{S} \quad \mathcal{P} = \mathcal{L}_\mathcal{P} \times \mathbb{N} \mid \mathcal{V}_\mathcal{P}\end{aligned}$$

Here, all sans-serif *symbols and the symbols of $\mathcal{F}$ are* constructors.

The set $\mathcal{A}$ contains both the *concrete* agent names ($\mathcal{L}_\mathcal{A} \times \mathbb{N}$) and the *variables* for agent names ($\mathcal{V}_\mathcal{A}$). The concrete agent names consist of a label and a natural number. The labels are for the interplay of Isabelle and the automated methods, for now it is sufficient to think just of an infinite set of agents, indexed by natural numbers. Similarly, $\mathcal{N}$, $\mathcal{S}$, and $\mathcal{P}$ define infinite reservoirs of concrete constants and variables for nonces, symmetric keys, and public keys.

For convenience, we write in the examples of this paper simply a, b, or i for concrete agent names, A, B for agent variables, n, n_1, etc. for concrete nonces, N_A, N_B etc. nonce variables. In

general, we use lower-case letters for constants and function symbols and upper-case letters for variables.

We distinguish atomic messages and composed messages (first and second line in the definition of $\mathcal{M}$). Except for the *untyped variables* of $\mathcal{V}_\mathcal{U}$, all atomic messages are of a particular type, namely from one of the sets $\mathcal{A}$, $\mathcal{N}$, $\mathcal{S}$, or $\mathcal{P}$. The constructors like agent ensure that the respective subsets of the message space are disjoint, for instance, no agent name can be a nonce. We discuss the details of typing below. In examples, we will omit the constructors for convenience, when the type is clear from the context.

Messages can be composed with one of the following operations: crypt and scrypt represent asymmetric and symmetric encryption, respectively. In both cases, the key (first argument) can be any message, allowing for composed keys and expressions like as $\mathsf{pk}(A)$ as keys. For convenience, we also use the common notation and write $\{m\}_k$ instead of crypt k m, and $\{|m|\}_k$ instead of scrypt k m. $\mathsf{inv}(M)$ represents the private key belonging to a public key, where we allow arbitrary message terms as an argument (as in $\mathsf{inv}(\mathsf{pk}(A))$). The concatenation of a list of messages is denoted by cat. In the examples, we omit the cat and the list brackets. We can apply any of the given function symbols of $\mathcal{F}$ to a message.

Consider for instance the concrete message $\{na_3, b\}_{\mathsf{pk}(a)}$ in our convenient notation; technically this may look as follows (for a suitable labeling and numbering):

$$\mathsf{crypt}\ (\mathsf{pk}\ (\mathsf{agent}\ (\mathsf{honest}, 1)))\ (\mathsf{cat}\ [\mathsf{nonce}\ (\mathsf{na}, 3), \mathsf{agent}\ (\mathsf{dishonest}, 2)])\ .$$

As we can see here, we have used two labels, honest and dishonest, for agents. This represents the default abstraction for agents that we use in the abstract model. We assume that whatever abstraction is chosen for the agents must be a refinement of this standard abstraction. This allows us to distinguish honest and dishonest agents throughout the reference model.

The inductively defined datatype is interpreted like a term in the free term algebra, i.e. syntactically different terms are interpreted as being different. We do not consider any algebraic properties of messages in this paper.

We use the standard notion of matching messages: a *ground* message s (i.e. one that contains no variables) *matches* a message t if and only if there is a substitution σ of all variables of t such that $s = t\sigma$. The constructors like agent here enforce a typing regime: typed variables can only be matched with atomic messages of the same type. Note that only untyped variables can be matched with composed messages. In rules for honest agents, we only used typed variables. Such a typed model which is standard in protocol verification, even in interactive verification with Isabelle [23,3], considerably simplifies the verification task. The typing can be justified by tagging as in [17].

2.2 Events and Traces

We define the set of events again as an inductive datatype, based on messages:

Definition 2. *The* set of events *is defined as follows:*

$$\mathcal{E} ::= \mathsf{iknows}\ \mathcal{M} \mid \mathsf{state}\ \mathcal{R}\ [\mathcal{M}] \mid \mathsf{secret}\ \mathcal{A}\ \mathcal{M} \mid \mathsf{witness}\ \mathcal{A}\ \mathcal{A}\ \mathcal{I}\ \mathcal{M} \mid \mathsf{request}\ \mathcal{A}\ \mathcal{A}\ \mathcal{I}\ \mathcal{M} \mid \mathsf{attack}\ \mathcal{M}$$

where $\mathcal{I}$ is a finite set of identifiers disjoint from all other symbols so far and $\mathcal{R} \subset \mathcal{I}$. A trace *is a finite sequence $\langle\, e_1 \,\#\, \cdots \,\#\, e_n \,\rangle$ of events e_i.*

The identifier set $\mathcal{I}$ contains constant symbols for the protocol variables, allowing us to describe as which protocol variable an agent interprets a particular message. We use Gothic fonts for identifiers, e.g. $\mathfrak{A}$ and $\mathfrak{B}$.

$$\frac{t \in \mathbb{T} \quad \mathsf{iknows}\ m \in [t] \quad \mathsf{iknows}\ k \in [t]}{\mathsf{iknows}\ \{m\}_k \# t \in \mathbb{T}} \qquad \frac{t \in \mathbb{T} \quad \mathsf{iknows}\ m \in [t] \quad \mathsf{iknows}\ k \in [t]}{\mathsf{iknows}\ \{|m|\}_k \# t \in \mathbb{T}}$$

$$\frac{t \in \mathbb{T} \quad \mathsf{iknows}\ m_1 \in [t] \quad \ldots \quad \mathsf{iknows}\ m_n \in [t]}{\mathsf{iknows}\ [m_1, \ldots, m_n] \# t \in \mathbb{T}} \qquad \frac{t \in \mathbb{T} \quad \mathsf{iknows}\ m \in [t]}{\mathsf{iknows}\ f(m) \# t \in \mathbb{T}}\ f \in \mathcal{F}_{pub}$$

$$\frac{t \in \mathbb{T} \quad \mathsf{iknows}\ \{m\}_k \in [t] \quad \mathsf{iknows}\ \mathsf{inv}(k) \in [t]}{\mathsf{iknows}\ m \# t \in \mathbb{T}}$$

$$\frac{t \in \mathbb{T} \quad \mathsf{iknows}\ \{m\}_{\mathsf{inv}(k)} \in [t] \quad \mathsf{iknows}\ k \in [t]}{\mathsf{iknows}\ m \# t \in \mathbb{T}}$$

$$\frac{t \in \mathbb{T} \quad \mathsf{iknows}\ \{|m|\}_k \in [t] \quad \mathsf{iknows}\ k \in [t]}{\mathsf{iknows}\ m \# t \in \mathbb{T}}$$

$$\frac{t \in \mathbb{T} \quad \mathsf{iknows}\ m_1 \in [t] \quad \ldots \quad \mathsf{iknows}\ m_n \in [t]}{\mathsf{iknows}\ m_1 \#, \ldots \# \mathsf{iknows}\ m_m \# t \in \mathbb{T}}$$

$$\frac{t \in \mathbb{T} \quad \mathsf{secret}\ A\ M \in [t] \quad \mathsf{iknows}\ M \in [t] \quad \mathsf{honest}\ A}{\mathsf{attack}\ M \# t \in \mathbb{T}}$$

$$\frac{t \in \mathbb{T} \quad \mathsf{request}\ B\ A\ id\ M \in [t] \quad \mathsf{witness}\ A\ B\ id\ M \notin [t] \quad \mathsf{honest}\ A}{\mathsf{attack}\ M \# t \in \mathbb{T}}$$

Fig. 2. The protocol independent rules of the reference model: the first four are *composition rules (C)*, the next four are *decomposition rules (D)* and the last two are *attack rules (A)*.

The event $\mathsf{iknows}\ m$ means that the intruder just learned the message m. The event $\mathsf{state}\ \mathfrak{R}\ msgs$ means that an honest agent playing role $\mathfrak{R}$ has reached a state of its protocol execution that is characterized by the list $msgs$ of messages. We need the other four events for expressing the goals in a protocol independent form when we introduce attack rules below.

2.3 Rules and Protocols

Based on these definitions, we can finally define protocols by a set of inductive rules on traces that have the following form:

$$\frac{t \in \mathbb{T} \quad \phi(t, e_1, \ldots, e_n)}{e_1 \# \ \ldots \ \# e_n \# t \in \mathbb{T}}$$

i.e. whenever t is a valid trace of the set $\mathbb{T}$ of traces and $e_1, \ldots, e_n$ are events that fulfill a certain condition ϕ with t, then also the extension of t with these events is also part of $\mathbb{T}$. Also, we have the rule that the empty trace is part of $\mathbb{T}$. Note that we require that all transition rules of honest agents contain only typed variables, i.e. no variables of $\mathcal{V}_{\mathcal{U}}$.

As an example, Fig. 2 shows the protocol independent rules for the intruder (C) and (D), following the standard Dolev-Yao style intruder deduction, as well as the attack rules (A), which follow the standard definitions of attacks in AVISPA [1]; here $[t]$ denotes the set of events in the trace t and $\mathcal{F}_{pub} \subseteq \mathcal{F}$ is the set of functions that is accessible to the intruder. The intruder rules are self-explanatory, e.g. if the intruder knows messages m and k on a trace t, then he can encrypt

m with k, and thus obtain $\{m\}_k$. Before we explain the attack rules, however, we describe the transition rules of role Alice of the standard example protocol, NSL [18] (more interesting examples are found in Sect. 6):

Example 1.

$$\frac{t \in \mathbb{T} \quad NA \notin \mathit{used}\ (t)}{\mathsf{iknows}\ \{NA, A\}_{\mathsf{pk}(B)} \# \mathsf{state}\ \mathfrak{A}\ [A, B, NA] \# \mathsf{witness}\ A\ B\ \mathfrak{N}_{\mathfrak{A}}\ NA \# \mathsf{secret}\ B\ NA \# t \in \mathbb{T}}$$

$$\frac{t \in \mathbb{T} \quad \mathsf{state}\ \mathfrak{A}\ [A, B, NA] \in [t] \quad \mathsf{iknows}\ \{NA, NB, B\}_{\mathsf{pk}(A)} \in [t]}{\mathsf{iknows}\ \{NB\}_{\mathsf{pk}(B)} \# \mathsf{request}\ A\ B\ \mathfrak{N}_{\mathfrak{B}}\ NB \# t \in \mathbb{T}}$$

Here, *used* (t) is the set of all atomic messages that occur in t to allow for the fresh generation of nonces. □

The goals of a protocol are described negatively by what counts as an attack. This is done by *attack rules* that have the event attack on the right-hand side. We now explain the events that we use in attack rules. First, secret $A\ M$ means that some honest agent (not specified) requires that the message M is a secret with agent A. Thus, it counts as an attack, if a trace contains both iknows M and secret $A\ M$ for an honest agent A (first attack rule in Fig. 2).

For authentication, the event witness $A\ B\ id\ m$ means that for a particular purpose identified by the identifier id, the honest agent A wants to transmit the message M to agent B. Correspondingly, when B believes to have received message M for purpose id from agent A, the event request $B\ A\ id\ M$ occurs. It thus counts as an attack, if request $B\ A\ id\ M$ occurs in a trace for an honest agent A and the trace does not contain the corresponding event witness $A\ B\ id\ M$ (second attack rule in Fig. 2. We call a trace an *attack trace* if it contains the attack event.

Definition 3. *Let a protocol be described by an inductive set R of rules. The protocol is said to be* safe*, if the least set $\mathbb{T}$ of traces that is closed under R contains no attack trace.*

Despite some differences, our model is similar to the one of Paulson [23]. He uses no explicit events for the state of honest agents, and rather characterizes their states by the messages they have sent and received. That is similar to our state terms which basically contain the sent and received messages. In fact, our model has the same persistence property as Paulson's: any rule of an honest agent can fire any number of times (with fresh nonces). This is because the event that an agent has reached a particular state of its protocol execution remains on the trace and there is no condition in the rules that requires that the agent has not progressed further. Like Paulson, we do not consider such conditions. More details on this phenomenon can be found in [20]. Further, Paulson uses events to express what different agents have said (and to whom). We use instead the iknows· events to express sent and received messages: the intruder is identified with the insecure communication medium, but this is basically the same communication model. Paulson does not consider any goal-related events, but rather encodes the goals directly on the exchanged messages (which is protocol dependent and can problematic for some protocols [20]).

3 Limiting Intruder Composition

The closure of the intruder knowledge under the composition rules of the intruder is generally infinite, e.g. the intruder can concatenate known messages arbitrarily. However, many of these messages are useless to the intruder since, due to typing, no honest agent accepts them. It is therefore intuitive that we do not loose any attacks if we limit intruder composition to terms, and subterms thereof, that some honest agent can actually receive. (A similar idea has been considered

e.g. in [25].) This limitation on intruder composition makes our approach significantly simpler, and we have therefore chosen to integrate this simplification into our reference model for this first version, and leave the generalization to an unlimited intruder for future versions. We formally define the transformation that limits intruder composition as follows:

Definition 4. *For a set of rules R that contain no untyped variables $\mathcal{V}_\mathcal{U}$, let M_R be the set of all messages that occur in an* iknows *or* secret *event of any rule of R along with as their sub-messages. We say that* an intruder composition rule r can compose terms for R*, if the resulting term of r can be unified with a term in M_R. In this case we call $r\sigma$ an* r-instance for compositions of M_R *if σ is the most general unifier between the resulting term of r and a message in M_R. We say that R is* saturated *if it contains all r-instances for composition in M_R.*

Since we excluded untyped variables, atomic messages in M_R are typed, i.e. of the form $\mathtt{type}(\cdot)$. Also, due to typing, every finite R has a finite saturation.

Theorem 1. *Given an attack against a protocol described by a set of rules $R \cup C \cup D \cup A$ where C and D are the intruder composition and decomposition rules and A are the attack rules. Let R' be a saturated superset of R. Then there is an attack against $R' \cup D \cup A$.*

Proof. Given an attack trace for $R \cup C \cup D \cup A$, we represent the attack trace in a tree form where the root is the attack-event, every node is labeled by the rule that created the event and has as successors the respective events of the trace that were required in the rule application. We show how to obtain a similar attack that does not use C rules but instead appropriate rules from R'.

First, we can easily exclude some pointless steps where the intruder for instance decomposes a message he has composed himself. In general, when the derivation tree of any event e properly contains itself another derivation of e, we can simply replace the outer subtree for e by the inner one and still have a valid attack. So we can safely assume that any derivation of e does not properly contain a derivation of e.

Now consider a node that is labeled with a rule of C and such that no other rule on the path to the root is a C rule and let $\mathsf{iknows}\ m$ be the resulting composed term. We show that there is a rule of R' that also can produce m from the respective subtrees, so repeated application of this argument gives a proof tree that contains no C-rules anymore. To that end, we show that m matches a subterm m' of an iknows event in R. It then follows that $m' \in M_R$ and, since m is the result of a composition and m' is typed, there is a corresponding instance of R' that produces m from the same assumptions, so we can replace the C rule with an R' rule.

To show that m matches some $m' \in M_R$, we consider the following cases. The considered C-node cannot be the root itself, because no rule of C produces the attack event. So the node has a parent node which is either an R, D, or A node. For the R-parent case, we immediately have the match.

For a D-parent, m necessarily corresponds to a decryption key: if it is an encryption that is deciphered or a pair decomposed, then we have a trivial composition-decomposition excluded above. Since there is no composition that produces private keys, i.e. that have $\mathsf{inv}(\cdot)$ as a result, m can only be the decryption key for a symmetric encryption or a signature, i.e. a proper subterm of the term t being analyzed. t can only be the result of an D or an R rule (C would lead to encryption-decryption, A produces no iknows event). In the case of an R-rule, due to typing, it has a subterm $m' \in M_R$ that matches m. In the case of a D-rule, we can trace the analyzed term t back to some super-term t' of t that is not obtained by decomposition, thus by an R rule. Thus t' is typed has a subterm m' that matches m.

For an A-parent, only the attack rule for secrecy has an $\mathsf{iknows}\ M$ event, and here M must occur in the form $\mathsf{secret}\ A\ M$ for some agent A. The secret event in the derivation can only result form an R rule, and thus a matching message is in M_R again. □

4 The Abstract Protocol Model

We now summarize two kinds of over-approximations of our model that are used in our automated analysis tool to cope with the infinite set of traces induced by the reference model. These techniques are quite common in protocol verification and a more detailed description can be found in [20]; we discuss them here only as far as they are relevant for our generation of Isabelle proofs. The first technique is a data abstraction that maps the infinite set of ground atomic messages (that can be created in an unbounded number of sessions) to a finite set of equivalence classes in the style of abstract interpretation approaches. The second is a control abstraction: we forget about the structure of traces and just consider *reachable events* in the sense that they are contained in some trace of the reference model. Neither of these abstractions is safe: each may introduce false attacks (that are not possible in the reference model). Also, it is not guaranteed in general that the model allows only for a finite number of reachable events, i.e. the approach may still run into non-termination.

4.1 Data Abstraction

In the style of abstract interpretation, we first partition the set of all ground atomic messages into finitely many equivalence classes and then work on the basis of these equivalence classes. Recall that atomic ground messages are defined as a pair (l, n) where l is a label and n is a natural number. We use the label to denote the equivalence class of the message in the abstraction. The abstract model thus identifies different atoms with the same label, and hence we just omit the second component in all messages of the abstract model.

There is a large variety of such data abstractions. For the proof generation the concrete way of abstraction is actually irrelevant, and we just give one example for illustration:

Example 2. The initial abstraction that OFMC uses for freshly created data is the following. If agent a creates a nonce n for agent b, we characterize the equivalence class for that nonce in the abstract model by the triple (n, a, b). This abstraction can be rephrased as "in all sessions where a wants to talk to b, a uses the same constant for n (instead of a really fresh one)". For a large number of cases, this simple abstraction is sufficient; in the experiments of Sect. 6, only two examples (NSL and Non-reversible Functions) require a more fine-grained abstraction. Note that OFMC automatically refines abstractions when the verification fails, but this mechanism is irrelevant for the proof generation. The protocol from Example 1, rules for A, then look as follows:

$$\frac{t \in \mathbb{T}}{\begin{array}{l}\mathsf{iknows}\ \{(\mathfrak{N}_{\mathfrak{A}}, A, B), A\}_{\mathsf{pk}(B)} \# \mathsf{state}\ \mathfrak{A}\ [A, B, (\mathfrak{N}_{\mathfrak{A}}, A, B)] \# \\ \mathsf{witness}\ A\ B\ \mathfrak{N}_{\mathfrak{A}}\ (\mathfrak{N}_{\mathfrak{A}}, A, B) \# \mathsf{secret}\ B\ (\mathfrak{N}_{\mathfrak{A}}, A, B), \# t \in \mathbb{T}\end{array}}$$

$$\frac{t \in \mathbb{T} \quad \mathsf{state}\ \mathfrak{A}\ [A, B, NA] \in [t] \quad \mathsf{iknows}\ \{NA, NB, B\}_{\mathsf{pk}(A)} \in [t]}{\mathsf{iknows}\ \{NB\}_{\mathsf{pk}(B)} \# \mathsf{request}\ A\ B\ \mathfrak{N}_{\mathfrak{B}}\ NB \# t \in \mathbb{T}}$$

Here, we only abstract NA in the first rule where it is created by the A. It is crucial that the nonce NB is not abstracted in the second rule: since NB is not generated by A, A cannot be sure a priori

that was indeed generated by B. In fact, if we also abstract NB here, the proof generation fails, because the resulting fixed-point does no longer over-approximate the traces of the reference model. More generally, fresh data are abstracted only in a rule where they are created. Finally, observe that the condition is gone that tells that the freshly created NA never occurred in the trace before, because now agents may actually use the same value several times in place of the fresh nonce.

The key idea to relate our reference model with the abstract one is the use of labels in the definition of concrete data. Recall that each concrete atomic message in the reference model is a pair of a label and a natural number. The finite set of labels is determined by the abstraction we use in the abstracted model; in the above example, we use $\mathcal{L}_{\mathcal{N}} = \{\mathfrak{N}_{\mathfrak{A}}, \mathfrak{N}_{\mathfrak{B}}\} \times \mathcal{L}_{\mathcal{A}} \times \mathcal{L}_{\mathcal{A}}$ where $\mathcal{L}_{\mathcal{A}}$ is the abstraction of the agents (for instance $\mathcal{L}_{\mathcal{A}} = \{(\mathfrak{honest}, \mathfrak{dishonest})\}$). As the atomic messages consist of both such a label and a natural number, the reference model is thus endowed with an infinite supply of constants for each equivalence class of the abstract model. The relationship between data in the reference and abstract models is straightforward: the abstraction of the concrete constant (l, n) is simply l. Vice-versa, each equivalence class l in the abstract model represents the set of data $\{(l, n) \mid n \in \mathbb{N}\}$ in the reference model.

It is crucial that in the reference model, the labels are merely annotations to the data and the rules do not care about these annotations, except for the distinction of honest and dishonest agents as discussed before. The labels however later allow us to form a security proof in the reference model based on the reachable events in the concrete model.

We need to take the abstraction into account in the reference model when creating fresh data. In particular, we need to enforce the labeling that reflects exactly the abstraction. We extend the assumptions of the first rule from Example 1 by a condition on the label of the freshly created NA:

$$\frac{t \in \mathbb{T} \quad NA \notin \mathit{used}\ (t) \quad \mathit{label}(NA) = (\mathfrak{N}_{\mathfrak{A}}, A, B)}{\begin{array}{c}\mathsf{iknows}\ \{NA, A\}_{\mathsf{pk}(B)} \# \mathsf{state}\ \mathfrak{A}\ [A, B, NA] \# \\ \mathsf{witness}\ A\ B\ \mathfrak{N}_{\mathfrak{A}}\ NA \# \mathsf{secret}\ B\ NA \# t \in \mathbb{T}\end{array}}$$

where $\mathit{label}(l, n) = l$. (Recall that every concrete value is a pair of label and a natural number.)

4.2 Control Abstraction

We now come to the second part of the abstraction. Even with the first abstraction on data, the model gives us an infinite number of traces (that are of finite but of unbounded length). The idea for simplification is that under the data-abstraction, the trace structure is usually not relevant anymore. In the reference model, we need the trace structure for the creation of fresh data and for distinguishing potentially different handling of the same constant in different traces. Under the data-abstraction, however, all these occurrences fall together. In the abstract model we thus abandon the notion of traces and consider only the set $\mathbb{E}$ of events that can ever occur.

Example 3. Our running example has now the following form:

$$\frac{}{\begin{array}{c}\mathsf{iknows}\ \{(\mathfrak{N}_{\mathfrak{A}}, A, B), A\}_{\mathsf{pk}(B)}, \mathsf{state}\ \mathfrak{A}\ [A, B, (\mathfrak{N}_{\mathfrak{A}}, A, B)], \\ \mathsf{witness}\ A\ B\ \mathfrak{N}_{\mathfrak{A}}\ (\mathfrak{N}_{\mathfrak{A}}, A, B), \mathsf{secret}\ B\ (\mathfrak{N}_{\mathfrak{A}}, A, B) \in \mathbb{E}\end{array}}$$

$$\frac{\mathsf{state}\ \mathfrak{A}\ [A, B, NA] \in \mathbb{E} \quad \mathsf{iknows}\ \{NA, NB, B\}_{\mathsf{pk}(A)} \in \mathbb{E}}{\mathsf{iknows}\ \{NB\}_{\mathsf{pk}(B)}, \mathsf{request}\ A\ B\ \mathfrak{N}_{\mathfrak{B}}\ NB \in \mathbb{E}}$$

Again, there are no guarantees that this construction terminates and gives a finite fixed-point or that the abstractions do not introduce attacks. We emphasize that a mistake in the tools and abstraction approach itself does not endanger the soundness of the entire verification approach presented here, as in the worst case the construction of the proof simply fails.

5 Turning Fixed-Points into Proofs

We now turn to the proof generator itself (see Fig. 1), putting the pieces together to obtain the security proof with respect to the reference model.

Let R_G denote in the following the given set of the reference model that describes the protocol, its goals, and the intruder behavior, as described in Sect. 2 and 3. Recall that OFMC chooses an abstraction for the data that honest agents freshly create, and refines these abstractions if the verification fails. As described in Sect. 4, the connection between the data abstraction and the reference model is made by annotating each freshly created message with a label expressing the abstraction. Since this annotation is never referred to in the conditions of any rule, the set of traces remains the same modulo the annotation. We denote by R_M the variant of the rules with the annotation of the freshly created data.

The next step is the inductive definition of the set of traces $\mathbb{T}$ in Isabelle, representing the least fixed-point of R_M. For such inductive definitions, Isabelle proves automatically various properties (e.g., monotonicity) and derives an induction scheme, i.e. if a property holds for the empty trace and is preserved by every rule of R_M, then it holds for all traces of $\mathbb{T}$. This induction scheme is fundamental for the security proof.

We define in Isabelle a set of traces $\mathbb{T}'$ that represents the over-approximated fixed-point FP computed by OFMC, expanding all abstractions. We define this via a *concretization function* $[\![\cdot]\!]$:

$$[\![l]\!] = \{(l, n) \mid n \in \mathbb{N}\} \qquad [\![f\ t_1\ \ldots\ t_n]\!] = \{f\ s_1\ \ldots\ s_n \mid s_i \in [\![t_i]\!]\}$$
$$[\![F]\!] = \cup_{f \in F} [\![f]\!] \qquad \mathbb{T}' = \{e_1 \# \ldots \# e_n \mid e_i \in [\![FP]\!]\}$$

This replaces each occurrence of an abstract datum with an element of the equivalence class it represents, and then builds all traces composed of events from the fixed-point. Note that while FP is finite, $\mathbb{T}'$ is infinite.

As the next step, the proof generation module proves several auxiliary theorems using a set of specialized tactics we designed for this purpose. Each proved auxiliary theorem can be used as a proof rule in subsequent proofs. We thus use Isabelle as a framework for constructing a formal tool in a logically sound way (see also [28]).

The first auxiliary theorem is that $\mathbb{T}'$ does not contain any attacks. The theorem is proved by unfolding the definition of the fixed-point and applying Isabelle's simplifier.

The main part of the proof generation is an auxiliary theorem for each rule $r \in R_M$ that $\mathbb{T}'$ is closed under r. In a nutshell, these theorems are also shown by unfolding the definition of $\mathbb{T}$ and applying the simplifier. In this case, however, on a more technical level, we need to convert the set comprehensions of the definition into a predicate notation so that the simplifier can recognize the necessary proof steps. This is actually also the point where the labels that annotate the abstraction silently fulfill their purpose: the rules are closed under *any* concretization of the abstract data with the elements from the equivalence class they represent. By our construction, the proof generation does not need to take care of the abstraction at all.

Protocol	FP	time [s]	Protocol	FP	time [s]
Andrew Secure RPC	113	1517	ISO three pass mutual	229	21448
Bilateral-Key Exchange	85	7575	NSCK	135	9471
Denning-Sacco	71	2549	NSL	75	117
ISO one pass unilateral	40	33	Non-Reversible Functions	196	21018
ISO two pass unilateral	56	77	TLS (simplified)	172	26982
ISO two pass mutual	104	442	Wide Mouthed Frog	87	1382

Table 1. Analyzing security protocols using Isabelle/OFMC

Using the theorems that all rules are closed under $\mathbb{T}'$, we can now show the last auxiliary theorem, namely that $\mathbb{T} \subseteq \mathbb{T}'$, i.e., that OFMC indeed computed an over-approximation of what can happen according to the reference model. This theorem is proved by induction, using the induction scheme we have automatically obtained from the definition of $\mathbb{T}$ above, i.e. we show that the subset relation is preserved for each rule of R_M, using the set of the auxiliary theorems.

Finally, we derive our main theorem that $\mathbb{T}$ contains no attack, which immediately follows from $\mathbb{T} \subseteq \mathbb{T}'$ and $\mathbb{T}'$ containing no attack. We have thus automatically derived the proof that the protocol is safe from the given reference model description and OFMC's output.

6 Experimental Results

For first experiments, we have considered several protocols from the Clark-Jacob library [10] and a simplified version of TLS. Tab. 1 shows the results in detail, namely the size of the fixed-point and the time to generate and check the Isabelle proof. Here, we have considered for each protocol all those secrecy and authentication goals that do actually hold (we do not report on the well-known attacks on some of these protocols that can be detected with OFMC). The runtime for generating the fixed-point in OFMC is negligible ($< 2s$ for each example), while the runtime for Isabelle/OFMC quickly increases with the size of the fixed-point, though it is feasible in all cases. We have set one day of runtime as limit; the proof generation in Isabelle/OFMC has exceeded this limit for some complex protocols like Kerberos (OFMC computed the fixed-point within minutes). We hope to improve on the proof generator performance by fine-tuning and specializing the low-level proof tactics where we currently use generic ones of Isabelle.

We suggest, however, that the proof generation time does not affect the experimentation with OFMC such as testing different designs and variants of a newly designed protocol, because the proof generation is meant only as a final step when the protocol design has been fixed and verified with OFMC.

7 Related and Future Work

There is a large number of automated tools for protocol verification. [4,6,7,9] in particular are close to the method that is implemented in the new fixed-point module of OFMC: they are all based on an over-approximation of the search space as described in Sect. 4: the first over-approximation concerns the fresh data, following the abstract interpretation approach of [13] and the second concerns the

control structure, i.e., considering a set of reachable events rather than traces. We propose that the other over-approximation-based tools can similarly be connected to Isabelle as we did it for OFMC.

The work most closely related to ours is a recent paper by Goubault-Larrecq who similarly considers generating proofs for an interactive theorem prover from the output of automated tools [16]. He considers a setting where the protocol and goal are given as a set S of Horn clauses; the tool output is a set S_∞ of Horn clauses that are in some sense saturated and such that the protocol has an attack iff a contradiction is derivable. He briefly discusses two principal approaches to the task of generating a proof from S_∞. First, showing that the notion of saturation implies consistency of the formula and that the formula is indeed saturated. Second, showing the consistency by finding a finite model. He suggests that the first approach is unlikely to give a practically feasible procedure, and rather follows the second approach using tools for finding models of a formula.

In contrast, our work, which is closer to the first kind of approach, shows that this proof generation procedure does indeed work in practice for many protocols, comparable to the results of [16]. We see the main benefit of our approach in the fact that we can indeed use the output of established verification tools dedicated to the domain of security protocols. However, note that the work of [16] and ours have some major differences which makes results hard to compare. First, we consider a reference model where the protocol is modeled as a set of traces; the generated proofs are with respect to this reference model and all abstractions are merely part of the automatic tools. In contrast, [16] considers only one protocol model based on Horn clauses, close to the abstract model in our paper. Taking the soundness of all these abstractions for granted is a weakness of [16]. However, also our approach takes some things for granted, namely a strictly typed model and, based on this, specialized composition rules for the intruder. The typing is common in protocol verification and can be justified by a reasonable protocol implementation discipline [17]. The second assumption is justified by Theorem 1. Our next steps are concerned with lifting these two assumptions from our reference model, i.e. allowing for an untyped reference model with unbounded intruder composition. First experiments suggest that at least the unbounded intruder composition can be feasibly integrated into the proof generation procedure.

We note that some automated verification tools are based on, or related to, automated theorem provers, e.g. SATMC [2] generates Boolean formulae that are fed into a SAT-solver, and ProVerif [4] can generate formulae for the first-order theorem prover SPASS [27]. While there is some similarity with our approach, namely connecting to other tools including the subtle modeling issues, this goes into a different direction. In fact, these approaches additionally rely on both the correctness of the translation to formulae, and the correctness of the automated theorem prover that proves them. In contrast, we generate the proof ourselves (using the help of the Isabelle API) and let Isabelle check that proof.

Several papers such as [8,5,20,12]have studied the relationships between protocol models and the soundness of certain abstractions and simplifications in particular. For instance, [12] shows that for a large class of protocols, two agents (an honest and a dishonest one) is sufficient. Recall that we have used this as a standard abstraction of honest agents. While such arguments have thus played an important role in the design of the automated tool and its connection to the reference model, the correctness of our approach does not rely on such arguments and the question whether a given protocol indeed satisfies the assumptions. Rather, it is part of the Isabelle proof we automatically construct that the abstract model indeed covers everything that can happen in the reference model. The automated verifier may thus *try out* whatever abstraction it wants, even if it is not sound. Once again, in the worst case, the proof in Isabelle simply fails, if the abstraction is indeed unsound

for the given protocol, e.g., when some separation of duty constraints invalidate the assumptions of the two-agents abstraction.

This was indeed one of the main motivations of our work: our system does not rely on the subtle assumptions and tricks of automated verification. This also allows for some heuristic technique that extends the classical abstraction refinement approaches such as [11]. There, the idea is to start with a simple most abstraction, and when the automatic verification fails, to refine the abstraction based on the counter-example obtained. This accounts for the effect that the abstraction may lead to incompleteness (i.e., failure to verify a correct system), but it is essential that one uses only sound abstractions (i.e., if the abstract model is flawless then so is the concrete model). With our approach, we are now even able to try out potentially unsound abstractions in a heuristic way, i.e. start with abstractions that *usually* work (like the two-agent abstraction). If they are unsound, i.e. the Isabelle proof generation fails, then we repeat the verification with a more refined abstraction.

Isabelle has been successfully used for the interactive verification in various areas, including protocol verification [23,3]. These works are based on a protocol model that is quite close the our reference model (see Sect. 2). There are several works on increasing the degree of automation in interactive theorem provers by integrating external automated tools [15,26,14,24,19]. These works have in common that they integrate generic tools like SAT or SMT tools. In contrast, we integrate a domain-specific tool, OFMC, into Isabelle.

As further future work, we plan the development of additional Isabelle tactics improving the performance of the verification in Isabelle. Currently, the main bottleneck is a proof step in which a large number of existential quantified variables need to be instantiated with witnesses. While, in general, such satisfying candidates cannot be found efficiently, we plan to provide domain-specific tactics that should be able infer witnesses based on domain-specific knowledge about the protocol model. Finally, we plan to eliminate the limitations on the intruder inductions from the model explained in Sect. 3. While this limitation can be reasonably justified in many cases, the fact that our approach relies on it is a drawback, both in terms of efficiency and also theoretically. In fact, tools like ProVerif instead employ a more advanced approach of rule saturation that allow to work without the limitation. We plan to extend our approach to such representations of the fixed-point.

Acknowledgments. The work presented in this paper was partially supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" (`www.avantssar.eu`). We thank Luca Viganò for helpful comments.

References

1. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Hankes Drielsma, P., Héam, P.C., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In: CAV'05. Springer-Verlag (2005). URL `http://www.avispa-project.org`
2. Armando, A., Compagna, L.: SAT-based Model-Checking for Security Protocols Analysis. Int. J. of Information Security **6**(1), 3–32 (2007)
3. Bella, G.: Formal Correctness of Security Protocols. Springer-Verlag (2007)
4. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: CSFW'01, pp. 82–96. IEEE Computer Society Press (2001)
5. Blanchet, B.: Security protocols: from linear to classical logic by abstract interpretation. Information Processing Letters **95**(5), 473–479 (2005)

6. Boichut, Y., Héam, P.C., Kouchnarenko, O., Oehl, F.: Improvements on the Genet and Klay technique to automatically verify security protocols. In: AVIS'04, pp. 1–11 (2004)
7. Bozga, L., Lakhnech, Y., Perin, M.: Pattern-based abstraction for verifying secrecy in protocols. Int. J. on Software Tools for Technology Transfer **8**(1), 57–76 (2006)
8. Cervesato, I., Durgin, N., Lincoln, P.D., Mitchell, J.C., Scedrov, A.: A Comparison between Strand Spaces and Multiset Rewriting for Security Protocol Analysis. In: ISSS 2002, LNCS 2609, pp. 356–383. Springer-Verlag (2003)
9. Chevalier, Y., Vigneron, L.: Automated Unbounded Verification of Security Protocols. In: CAV'02, LNCS 2404, pp. 324–337 (2002)
10. Clark, J., Jacob, J.: A survey of authentication protocol: Literature: Version 1.0 (1997). `www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`
11. Clarke, E., Fehnker, A., Han, Z., Krogh, B., Ouaknine, J., Stursberg, O., Theobald, M.: Abstraction and counterexample-guided refinement in model checking of hybrid systems. Int. J. of Foundations of Computer Science **14**(4), 583–604 (2003)
12. Comon-Lundh, H., Cortier, V.: Security properties: two agents are sufficient. In: ESOP'2003, LNCS 2618, pp. 99–113. Springer-Verlag (2003)
13. Cousot, P.: Abstract interpretation. Symposium on Models of Programming Languages and Computation, ACM Computing Surveys **28**(2), 324–328 (1996)
14. Erkök, L., Matthews, J.: Using Yices as an automated solver in Isabelle/HOL. In: AFM'08 (2008)
15. Fontaine, P., Marion, J.Y., Merz, S., Nieto, L.P., Tiu, A.F.: Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In: H. Hermanns, J. Palsberg (eds.) TACAS, LNCS 3920, pp. 167–181. Springer-Verlag (2006)
16. Goubault-Larrecq, J.: Towards producing formally checkable security proofs, automatically. In: CSF'08, pp. 224–238. IEEE Computer Society (2008)
17. Heather, J., Lowe, G., Schneider, S.: How to prevent type flaw attacks on security protocols. In: CSFW'00. IEEE Computer Society Press (2000)
18. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: TACAS'96, LNCS 1055, pp. 147–166. Springer-Verlag (1996)
19. Meng, J., Quigley, C., Paulson, L.C.: Automation for interactive proof: First prototype. Information and Computation **204**(10), 1575–1596 (2006)
20. Mödersheim, S.: On the Relationships between Models in Protocol Verification. J. of Information and Computation **206**(2–4), 291–311 (2008)
21. Mödersheim, S., Viganò, L.: The open-source fixed-point model checker for symbolic analysis of security protocols. In: Fosad 2007–2008–2009, *LNCS*, vol. 5705, pp. 166–194. Springer-Verlag (2009)
22. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. LNCS 2283. Springer-Verlag (2002)
23. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. of Computer Security **6**(1-2), 85–128 (1998)
24. Paulson, L.C., Susanto, K.W.: Source-level proof reconstruction for interactive theorem proving. In: K. Schneider, J. Brandt (eds.) TPHOLs, LNCS 4732, pp. 232–245. Springer-Verlag (2007)
25. Roscoe, A.W., Goldsmith, M.: The perfect spy for model-checking crypto-protocols. In: DIMACS'97 (1997)
26. Weber, T., Amjad, H.: Efficiently checking propositional refutations in HOL theorem provers. J. of Applied Logic **7**(1), 26 – 40 (2009)
27. Weidenbach, C., Schmidt, R.A., Hillenbrand, T., Rusev, R., Topic, D.: System description: Spass version 3.0. In: CADE-21, pp. 514–520. Springer-Verlag (2007)
28. Wenzel, M., Wolff, B.: Building formal method tools in the Isabelle/Isar framework. In: K. Schneider, J. Brandt (eds.) TPHOLs 2007, LNCS 4732, pp. 352–367. Springer-Verlag (2007)