Report

On the relationships between models in protocol verification (extended version)

Author(s):
Mödersheim, Sebastian

Publication Date:
2011

Permanent Link:
https://doi.org/10.3929/ethz-a-006775802

Rights / License:
In Copyright - Non-Commercial Use Permitted
On the Relationships between Models in Protocol Verification  
(Extended Version)*

Sebastian Mödersheim

Information Security Group, Dep. of Computer Science, ETH Zurich, Switzerland
www.inf.ethz.ch/~moedersheim

Abstract

We formally investigate the relationship between several models that are widely used in protocol verification, namely variants of the inductive model of message traces inspired by Paulson’s approach, and models based on rewriting. More precisely, we prove several over-approximation relationships between models, i.e. that one model allows strictly more traces or reachable states than the other. This is common in verification: often an over-approximation is easier to prove correct than the original model, and proving the over-approximation is safe implies that the original model is safe—provided that the models are indeed in an over-approximation relation. We also show that some over-approximations are not sound with respect to authentication goals. The precise formal account that we give on the relation of the models allows us to correct the situation.

1 Introduction

Security protocols have been intensively studied using formal methods in the past 25 years, and a large variety of formalisms, models, and techniques have been developed in order to cope with the different kinds of infinity that arise in verification. Most notably, there is no bound on the number of sessions (protocol runs) that can be executed in parallel, and there are infinitely many messages an intruder can construct from what he has observed on the network, even assuming that he cannot break the cryptography. The variety of different models results in a number of problems: it is hard to tell whether the statements proved by different verification tools are equivalent, how tools can be combined, and whether a particular meta-argumentation about one model (e.g. that certain restrictions are without loss of generality) can be carried over to other models.

In this paper, we consider the relationships between variants of two widely used models. The first model is based on set rewriting, which is for instance employed as the input language for the verification tools that are part of the AVISPA-tool [2], the CAPSL framework [16] and [22]. The second model is inspired by Paulson’s approach of message traces based on the Isabelle theorem prover [24]. This latter model has influenced a number of automated verification approaches as discussed below.

Although we consider two particular models, the ideas presented in this paper are general. There are several works that argue for the equivalence of models in different formalisms [12] and

---

*This work was partially supported by the Zurich Information Security Center. This work represents the views of the author. The author also wishes to thank Luca Viganò for inspiring discussions and helpful comments.
the universality of their results [14]. However, it is not the focus of this paper to discuss all design choices and to compare with all existing formalisms. Rather we want to point out that most current protocol models follow the same basic idea as our set rewriting model: the protocol is a set of processes (the honest agents) each of which has a local state that is updated when receiving a message and sending an answer, and these processes are connected via an intruder-controlled network. And we thus see the set rewriting approach as a representative of the “standard” way to model protocols. In contrast, the message trace model has no explicit notion of processes or local state; rather we have rules that tell us how a given trace of exchanged messages can be extended with further messages.

The focus of this work is on the over-approximation relationship between two models, i.e. when one model allows strictly more traces or reachable states than the other. Over-approximations can induce attacks that are of type false positive, i.e. attacks that work only in the over-approximated model, but not in the original model. The false positives are thus in some sense artifacts created by the over-approximation. For falsification (i.e. detecting attacks) this is problematic, as the “real” attacks may be buried under false positives. On the other hand, for verification (i.e. trying to prove a protocol correct) over-approximation does make sense: given a precise model and an over-approximation of it, proving that the over-approximation is safe is often much easier than in the original model and immediately implies that the precise model is safe as well.

Our contributions are as follows (see also Fig. 1): We formally prove that the message trace model $T$ is an over-approximation of the set rewriting model $R$, i.e. it contains many traces that have no counter part in the set rewriting model (or in reality), while conversely all reachable states of the set rewriting model have their counter part in the message trace model. Moreover, we show that a certain variant $P$ of the set rewriting model is equivalent to the message trace model in a sense that is made precise in this paper, yielding a precise understanding of the relationship between the two models. This relation has never been expressed in the literature before.

We further show how the over-approximations inherent in $R$ and $T$ can be used to abstract away a large part of the control structure of protocols, and this control abstraction is the basis of many automated verification approaches [7, 9, 10, 11, 13, 18, 25]. We formally prove that these models ($F$ and $E$) are over-approximations of the message trace model, a fact which has also never been formally analyzed before.

These relationships yield a better understanding of these models. While it is immediate that the over-approximations of the different models are sound for reasoning about secrecy, it turns out that it is not sound in general for authentication goals, i.e. a protocol may have an authentication flaw in the set rewriting model, but appear flawless in the over-approximation. We attribute the discovery of this subtlety to the rigid formal study of the models and their relationship. We also show how to soundly encode authentication goals (based on data abstraction) in the models $F$ and $E$.

The work closest related to ours is [8] which considers two models similar to $P$ and $E$ and shows that every attack against secrecy in the original model can be recast in the over-approximation. It does not examine, however, all the relationships between models as we do, and does not consider authentication goals. Also, several papers like [23] deal with the safe over-approximation by data abstraction, which is however orthogonal to this paper.

Fig. 1 gives an overview of the protocol models we consider and the theorems about their relationship. We proceed as follows: after summarizing preliminaries in Sec. 2, we define the set rewriting model and the message trace model in Sec. 3. In Sec. 4 we formally show that the message trace model over-approximates the set rewriting model. In Sec. 5 we show how to use the over-approximation to perform a complete control abstraction. In Sec. 6 we consider the specification of security goals for the protocols in the different models, based on our results about their relationship. In Sec. 7 we conclude with a summary and an outlook on future work.
2 Preliminaries

The models we consider are all based on the standard intruder model by Dolev and Yao [17], where honest agents communicate over a hostile network controlled by a dishonest party, the intruder, and where the cryptography is assumed to be perfect, i.e. the intruder can decrypt an encrypted message only if he knows the proper key.

2.1 Messages as Terms

Messages are modeled as terms, where agent names, atomic keys and nonces are represented by constants and all (cryptographic) operations are represented by function symbols. For instance, the function symbol \( \{m\}_{k} \) represents the asymmetric encryption of message \( m \) with key \( k \). The pair of the messages \( m_1 \) and \( m_2 \) is written as \( m_1, m_2 \). Note that the pairing operator is not associative, although the notation may suggest so. Rather, \( m_1, m_2, m_3 \) stands for \( m_1, (m_2, m_3) \).

We introduce further symbols when needed below. Throughout this paper, \( \Sigma \) denotes a countably infinite set of constant and function symbols, and \( \Sigma_0 \subseteq \Sigma \) are the constant symbols, where \( i \in \Sigma_0 \) denotes the intruder. We assume that there is a distinguished subset \( \Sigma'_0 \) of \( \Sigma_0 \) for the creation of fresh data. \( \mathcal{V} \) denotes a countable set of variable symbols disjoint from \( \Sigma \). By convention, variable symbols of the term algebra begin with an upper-case letter and constant and function symbols with a lower case letter. (This does not hold for the meta-variables of our argumentation.) The set of terms built from \( \Sigma \) and a set \( \mathcal{V} \subset \mathcal{V} \) is denoted by \( T_{\Sigma}(\mathcal{V}) \), and we write \( T_{\Sigma} \) for \( \mathcal{V} = \emptyset \), and call it the set of ground terms. \( T_{\Sigma}(\mathcal{V}) \) is a \( \Sigma \)-algebra, usually called the free term-algebra. This means that any two terms \( t_1 \) and \( t_2 \) are interpreted equally in this algebra, denoted \( \approx \), iff they are syntactically equal, e.g. \( \{m\}_k \approx \{m'\}_{k'} \) iff \( k \approx k' \) and \( m \approx m' \). This reflects one aspect of the perfect cryptography assumption. The free-algebra interpretation of terms is standard in the formal analysis of security protocols, and except for the set rewriting operator introduced below, we stick with this assumption. We also use other standard concepts like substitution and matching, see e.g. [5].

2.2 Dolev-Yao Intruder Deduction

As it is standard, we inductively define the set of messages an intruder can deduce from a given set of ground messages \( IK \) ("intruder knowledge") as the least set that contains \( IK \) and that is closed under a number of rules, and denote it by \( \mathcal{DY}(IK) \). For instance, using a unary symbol \( \text{inv}(k) \) to denote the private key corresponding to public key \( k \), we can define the following rules for asymmetric encryption and decryption:

\[
\frac{k \in \mathcal{DY}(IK) \quad m \in \mathcal{DY}(IK)}{\{m\}_k \in \mathcal{DY}(IK)} \quad \frac{\{m\}_k \in \mathcal{DY}(IK) \quad \text{inv}(k) \in \mathcal{DY}(IK)}{m \in \mathcal{DY}(IK)}
\]

\[
\frac{\text{inv}(k) \in \mathcal{DY}(IK) \quad k \in \mathcal{DY}(IK)}{m \in \mathcal{DY}(IK)}
\]

Depending on the set of cryptographic operators, there may be many similar rules. Throughout this paper, we consider a fixed \( \mathcal{DY} \) deduction relation. We assume that the reader is familiar with these kinds of inductive definitions, see e.g. [1]; in particular, note that the least closure is uniquely
defined. For $DY$, using the least closure reflects the central aspect of the perfect cryptography assumption: the intruder cannot perform any deduction steps except those explicitly given by these rules.

### 2.3 Set Rewriting

A **rewrite rule with fresh constants** has the form $l = [V] \Rightarrow r$ for two terms $l$ and $r$ and a set of variables $V$, such that $\text{vars}(l) \cap V = \emptyset$ and $\text{vars}(l) \cup V \supseteq \text{vars}(r)$. ($\text{vars}(t)$ denotes the set of variable symbols that occur in $t$.) If $V = \emptyset$, we simply write $l \Rightarrow r$. The intuition is the following: given a term $t$, if a subterm matches the left-hand side of the rewrite rule, we can replace this occurrence with the right-hand side; the set of variables $V$ means that fresh constants (that do not appear in $t$) are created in this transition. This mechanism is used to create fresh constants during protocol execution. More formally, a set of rewrite rules $R$ induces a rewrite relation $\rightarrow_R$ on terms as follows: $t \rightarrow_R s$ iff there is a rule $l = [V] \Rightarrow r \in R$, a position $p$ in $t$ and a substitution $\sigma$ such that $t$ at position $p$ is equivalent to $l\sigma$, $\sigma$ maps each variable of $V$ to a different constant in $\Sigma_0$ that does not occur in $t$, and $s$ is obtained from $t$ by replacing the subterm of $t$ at position $p$ with $r\sigma$. Note that if $t$ is a ground term, then so are $l\sigma$, $r\sigma$ and $s$. For set rewriting we assume an operator “.” that is associative, commutative, and idempotent (ACI). These properties can be expressed by a set of equations modulo which terms are interpreted. One can use this operator to define state transition systems as follows. Every state is a finite set, composed by “.”, of facts that hold in this state. Each fact is expressed by ground terms of $T_\Sigma$, e.g. we use the fact symbol $\text{iknows}(m)$ to denote that the intruder knows the message $m$. Transitions from state to state are expressed by rewrite rules containing the “.” operator. For instance the rule

$$\text{iknows}(M).\text{iknows}(K) \Rightarrow \text{iknows}(M).\text{iknows}(K).\text{iknows}(\{M\}_K)$$

can be applied to any state, which contains as a subset facts that match with $\text{iknows}(M)$ and $\text{iknows}(K)$. (This could even be the same fact if for the matching substitution $\sigma$ it holds $M\sigma = K\sigma$.) This matched subset is replaced with the set of facts of the right-hand side under the matching substitution. Therefore this rule would add the respective encrypted message to the intruder knowledge; the effect that it only increases the set is achieved by repeating the left-hand side occurrence on the right-hand side. Since we will not have any rules where pieces of intruder knowledge are removed, we define the following syntactic sugar: for the special case of $\text{iknows}(\cdot)$ facts, every left-hand side occurrence is implicitly repeated on the right-hand side, so we do not have to repeat it there. Thus we may write the above rule as $\text{iknows}(M).\text{iknows}(K) \Rightarrow \text{iknows}(\{M\}_K)$.

### 3 Models Based on Set Rewriting and Message Traces

In order to be able to make a formal comparison of models, we must have a common basis for describing protocols. We choose a simple strand-space style formalism, which limits the class of protocols we can consider, but makes the presentation much easier.

#### 3.1 Message Patterns

The message pattern description of a protocol is given by a finite, non-empty set of role names $\mathcal{R}_1, \ldots, \mathcal{R}_r$ and for every role $\mathcal{R}$ a finite, non-empty list of message terms of $T_\Sigma(V)$, denoted by $m_0^R, \ldots, m_n^R$. We denote the last index $n$ by $\text{last}^R$ and require that it is even. We also require that for every role $\mathcal{R}$, the term $m_0^R$ is a non-empty concatenation of variables built with the pair operator. This term will later be instantiated with agent names, and the first component is the agent playing the $\mathcal{R}$. We thus denote with $\text{player}(m)$ the first component of the message $m$ when

---

1 The often considered multi-set rewriting requires only an AC operator, which is technically simpler while set rewriting leads to a smoother formalization.
The first appearance of different protocol runs. Finally, we assume a finite set of ground messages; a finite number of agents can be justified as in [14] by identifying the names of honest agents in protocol runs is finite; however, we do not limit the number of protocol runs. The restriction to (otherwise Theorem 3 would not hold), which means that the set of agents who can participate in

\(m\) is an instance of \(m^R\). We assume that none of the constants of \(\Sigma'_0\) (which are reserved for fresh data creation) are contained in the protocol description.

The message terms with an odd index stand for incoming messages, the ones with even index stand for outgoing messages. Intuitively, variables that first appear in \(m^R\), i.e. not in \(m^R_0, \ldots, m^R_{n-1}\), are learned if \(i\) is of odd index, and freshly created, if \(i\) is of even index greater than 0. The function \(\text{fresh}_n^R = \text{vars}(m^R_n) \setminus (\text{vars}(m^R_0) \cup \ldots \cup \text{vars}(m^R_{n-1}))\) abbreviates the set of fresh variables. Note that we do not require that the combination of the rules yields a meaningful protocol.

\textbf{Example.} The “canonical example” of the Needham-Schroeder Public Key Protocol with Lowe’s fix (NSL) can be described by the message pattern sequences displayed in Fig. 2. The layout is arranged so that corresponding messages of different roles are on the same line. Since role \(A\) is starting the protocol, the first incoming message is the dummy message start; similarly, as \(B\) has no outgoing message in the last step, we have here the dummy message end. These two dummy messages make the rest of the formalization uniform as we do not have to consider special cases for the first and last message. During the protocol execution, \(A\) freshly creates the nonce \(N_A\) (since the first appearance of \(N_A\) in any of \(A\)’s messages is outgoing) and learns two values \(PK_B\) and \(N_B\) (since these variables first appear in incoming messages for role \(A\)).

Observe that the message terms \(m^A_2\) and \(m^A_3\) are identical except for the subterms \(PK_B\) and \(pk(B)\), respectively. This reflects the fact that \(S\) knows everybody’s public key, but \(A\) does not. Thus \(A\) would accept any message at the position of \(PK_A\) and use it in subsequent communication. \(A\) only knows her own public/private key-pair and the public key of \(S\). In fact, the private key of \(A\) is never explicitly used, but it is implicitly used, since \(A\) expects to receive a message encrypted with its public key in \(m^A_5\).

In some cases, it is necessary to limit the set of agents that can play a certain role, for instance the server in the NSL protocol cannot be the intruder (otherwise the protocol could not guarantee anything). Thus, the protocol description also includes a predicate \(\text{inst}_R(m)\) which says whether \(m\) is an allowed instance of \(m^R\). We assume that the set of \(m\) with \(\text{inst}_R(m)\) is finite for every role (otherwise Theorem 3 would not hold), which means that the set of agents who can participate in protocol runs is finite; however, we do not limit the number of protocol runs. The restriction to a finite number of agents can be justified as in [14] by identifying the names of honest agents in different protocol runs. Finally, we assume a finite set of ground messages \(IK_0\) associated with the

<table>
<thead>
<tr>
<th>Role (A)</th>
<th>Role (B)</th>
<th>Role (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m^A_0)</td>
<td>(m^B_0)</td>
<td>(m^S_0)</td>
</tr>
<tr>
<td>(A, B, S)</td>
<td>(B, S)</td>
<td>(S)</td>
</tr>
<tr>
<td>(m^A_1)</td>
<td>(m^B_1)</td>
<td>(m^S_1)</td>
</tr>
<tr>
<td>(\text{start})</td>
<td>({N_A, A}_{PK_B})</td>
<td>(A, B)</td>
</tr>
<tr>
<td>(m^A_2)</td>
<td>(m^B_2)</td>
<td>(m^S_2)</td>
</tr>
<tr>
<td>(A, B)</td>
<td>(B, A)</td>
<td>({B, pk(B)}_{inv(pk(S))})</td>
</tr>
<tr>
<td>(m^A_3)</td>
<td>(m^B_3)</td>
<td>(m^S_3)</td>
</tr>
<tr>
<td>({B, PK_B}_{inv(pk(S))})</td>
<td>({A, PK_A}_{inv(pk(S))})</td>
<td>(A, B)</td>
</tr>
<tr>
<td>(m^A_4)</td>
<td>(m^B_4)</td>
<td>(m^S_4)</td>
</tr>
<tr>
<td>({N_A, A}_{PK_B})</td>
<td>({N_A, N_B, B}_{PK_A})</td>
<td>({A, pk(A)}_{inv(pk(S))})</td>
</tr>
<tr>
<td>(m^A_5)</td>
<td>(m^B_5)</td>
<td>(m^S_5)</td>
</tr>
<tr>
<td>({N_A, N_B, B}_{PK_A})</td>
<td>({N_B}_{pk(B)})</td>
<td>(A, B)</td>
</tr>
<tr>
<td>(m^A_6)</td>
<td>(m^B_6)</td>
<td>(m^S_6)</td>
</tr>
<tr>
<td>({N_B}_{pk(B)})</td>
<td>(\text{end})</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Example of message sequences: the Needham-Schroeder Public Key Protocol with Lowe’s fix (NSL).
**InitialState:**

\( \text{iknows}(m) \) for every \( m \in IK_0 \)

**Rules:**

\( \Rightarrow \text{state}_R(m) \) for every role \( R \) and message \( m \) such that \( \text{inst}_R(m) \)

\( \text{state}_R(m^R_0, \ldots, m^R_{n-2}).\text{iknows}(m^R_{n-1}) = [\text{fresh}^R_n] \Rightarrow \text{iknows}(m^R_n).\text{state}_R(m^R_0, \ldots, m^R_n) \)

for every role \( R \) and even index \( n \) such that \( 0 < n \leq \text{last}^R \)

\( \text{iknows}(m_1) \ldots \text{iknows}(m_n) \Rightarrow \text{iknows}(m) \)

for every intruder rule \( m_1 \in DY(IK) \ldots m_n \in DY(IK) \)

**Figure 3:** The set rewriting model. Note that \( \text{iknows}(\cdot) \) on a left-hand side is implicitly present on the right-hand side.

protocol, representing the initial knowledge of the intruder. For the example of the NSL protocol, \( IK_0 = \{i, pk(i), inv(pk(i)), s, pk(s), \text{start}\} \).

Since we never consider more than one protocol at a time, we consider the protocol description (consisting of the message patterns, initial intruder knowledge and allowed instantiations) as fixed for the rest of this paper, in order to avoid indexing everything with the protocol description. The formalization of goals for a protocol is addressed in Sec. 6, when the relationship of the models is clear. In the comparison of models, we focus on what the intruder can deduce and a notion of the state of honest agents.

### 3.2 The Set Rewriting Model

Fig. 3 defines the set rewriting model by an initial state and a set of set rewrite rule schemata (i.e. rules that are parameterized over the protocol description); an example is given in Fig. 5. The initial state for the set rewriting model consisting of an \( \text{iknows}(\cdot) \) fact for every message that the intruder initially knows. The first rule schema (which has no left-hand side and can therefore be applied to any state) describes the creation of a new protocol run: for every role and for every permitted instantiation of the agent names of that role, we can create a new local state of an agent. The binary fact symbol \( \text{state}_R(m) \) represents the local state of an agent in the execution of the protocol in \( R \); the information that the agent has gathered so far is a concatenation of the instantiations (i.e. his own name and maybe other predetermined agents) and the messages he has sent and received up to now. The next rule schema describes the transitions of honest agents: when an agent is at the stage to have sent the message \( n - 2 \) of his role description, and receives \( n - 1 \) from the network (i.e. the intruder), then he can answer with message \( n \) to the network (i.e. the intruder) and update his local state by adding message \( n - 1 \) and \( n \). (Recall that the state-fact of the left-hand side is removed, while the knows-fact remains due to the syntactic sugar we have defined.) Also, in such a transition all fresh variables (\( \text{fresh}^R_n \)) are substituted for constants that have not occurred so far. The last rule schema directly expresses the deduction rules of the intruder. We now inductively define the set of reachable states \( R \) as the least set of states that includes the initial state and that is closed under the \( \rightsquigarrow_R \) relation induced by the rules.

While we store the entire message terms sent and received in agent state facts, many rewriting approaches rather just store the set of variables that occur in these messages, in order to have smaller state-facts which may be an important efficiency issue. Note that in this case the subterm in several steps of the protocol execution may not change since no new incoming and outgoing messages are handled. Therefore one must in these cases add a step number the list of variables to distinguish different states of the protocol execution.
the behavior of honest agents, we have a rule for each role $R$:

\[
\begin{array}{c}
\text{if } t \in T \quad A \in \Sigma_0 \setminus \Sigma'_0 \quad M \in \mathcal{DY}(IK_0 \cup \{M \mid \text{snd}_A(M) \in [t]\})
\end{array}
\]

\[
\frac{t \in T \quad \text{inst}_R(M_0)}{\text{fresh}_R^A \subseteq \Sigma_0 \setminus \text{used}(t) \quad \text{rev}_A(M_1^R) \in [t] \quad \text{snd}_A(M_2^R) \in [t] \quad \ldots \quad \text{rev}_A(M_{n-1}^R) \in [t]}
\]

\[
\text{snd}_A(M_n^R) : t \in T
\]

for every $R$, for every $n$ that is even and $0 < n \leq \text{last}^R$, and $A = \text{player}(m_0^R).

Figure 4: The Message Trace Model

The reader may wonder whether there can be a problem due to the fact that in set rewriting we cannot have multiple identical state-facts. However, we can at any time use the initialization rule to obtain the initial state facts, and from there repeat any execution we had before up to the point where the agent creates fresh data. From that point on, the state facts are already different due to the fresh data. This argumentation allows us not to consider session identifiers. (However, when there is not such an instantiation rule, but we want to define a finite number of sessions by respective state facts in the initial state, e.g. as in the Intermediate Format of the AVISPA-tool [4], then such session identifiers are indeed necessary.)

3.3 The Message Trace Model

The message trace model characterizes the protocol by a set of traces $T$ that is defined as the least set closed under the rules in Fig. 4; an example is given in Fig. 5. A trace is a finite sequence of events. In our model, there are two kinds of events, namely $\text{snd}_a(m)$ and $\text{rev}_a(m)$, where $a$ is a constant and $m$ is a ground term, representing that $a$ sends or receives the message $m$. The first rule simply says that the empty trace $<>$ is in $T$. The second rule formalizes that the intruder can see all sent messages on the network, and that all messages, which any agent $A$ receives, are chosen by the intruder from the set of messages he can deduce from what he has seen on the network. We allow as an agent name any constant of $\Sigma_0 \setminus \Sigma'_0$ since there is no notion of typing. Recall that $\Sigma'_0$ is the distinguished subset of $\Sigma_0$ reserved for creating fresh data. The symbol $\;^*\;$ represents concatenating an event to a trace, and $[t]$ gives the set of events of a trace. To describe the behavior of honest agents, we have a rule for each role $R$ and suitable index $n$: if a trace containing all send and receive events up to point $n$ in the protocol execution of role $R$, then the trace can be extended by the next send event of $R$. Here, $\text{used}(t)$ is the set of constants that occur in any message of trace $t$. This formalization is close to the one of [24], but there are some differences.

One difference is that we do not consider a typed model here, as this question is orthogonal to our results, which can be seen as follows.\footnote{A typed model (in the context of security protocols) means that the set of constants $\Sigma_0$ is partitioned into several subsets representing the different types, e.g. agent names, nonces, public-keys etc. Variables of the rules may also be assigned a type, meaning that they can only be instantiated with constants from the respective subset of $\Sigma_0$. In particular, a variable cannot be replaced with a composed term. For instance, in the NSL example, if the intruder sends some arbitrary term that he has composed for a message part of type nonce, then in the typed model this will not be accepted—although in reality we can hardly argue that an agent can distinguish a random number from some composed term (e.g. an encryption of a random number) if they have the expected length in bits. However, one can argue that for a large class of protocols, when using a particular way of implementing the protocol, the typing is not really a restriction [20]. This particular way of typed models is only used in the analysis of security protocols, and significantly deviates from type systems of functional programming languages.} We can implement a typed model by using unary function symbols for types. E.g. $\text{nonce}(c)$ for $c \in \Sigma_0$ would represent a typed constant, and $\text{nonce} N$ for $N \in \mathcal{V}$ a typed variable. For some variables, e.g. the variables $PK_A$ in the NSL example, so-called composed types are required, in this case $pk(\text{agent}(A))$ to reflect the term structure of a well-typed message, see [4]. Given a protocol description in the message sequence model, we can enforce typing in this protocol by the following transformation: we replace all constants with typed
The second difference is that [24] uses a combination of send and receive events, denoted \( a \rightarrow b : m \), which at first sight looks like a synchronous communication model (where, in particular, the intruder cannot intercept messages), but this is not the case. In recent works, similar events as in our model are used, called Says and Gets [6].

There are further minor differences: in [24], we have two functions \( \text{synth} \) and \( \text{analz} \) that represent each a part of the \( \text{DY} \) closure, namely the synthesis and analysis rules, and instead of \( \text{DY}(IK) \) the closure \( \text{synth}(\text{analz}(IK)) \) is employed, which in general is a proper subset of \( \text{DY}(IK) \) (only when the protocol uses only atomic keys is executed in a typed model, these closures are equal). We thus use the complete \( \text{DY}(IK) \). Also, the way we use the set of permitted instances of agent names is not present in the rules of [24]; this is due to the fact that it does not need to be repeated in each step when given a concrete protocol, but necessary when we have an arbitrary protocol as a message pattern.

### 4 The Over-Approximation and Persistent Set Rewriting

The intuition about why the models are so similar is that every state fact of the set rewriting model in some way corresponds to the respective send and receive events in the message trace models. We formalize this intuition by a function \( [\cdot] \) that maps a state fact to a set of events:

\[
[\text{state}_B((B, S), \{N_A, A\}_{pk(B)}, (B, A))).\text{iknows}(\{A, PK_A\}_{\text{inv}(pk(S))}) = [N_B] \Rightarrow \text{iknows}(\{N_A, N_B, B\}_{PK_A}).\text{state}_B(\ldots, \{A, PK_A\}_{\text{inv}(pk(S))}, \{N_A, N_B, B\}_{PK_A})
\]

\[
t \in T \quad \begin{array}{c}
N_B \in \Sigma_0 \setminus \text{used}(t) \quad \text{rcv}_B(\{N_A, A\}_{pk(B)}) \in [t] \\
\text{snd}_B((B, A)) \in [t]
\end{array} \quad \text{rcv}_B(\{A, PK_A\}_{\text{inv}(pk(S))}) \in [t]
\]

\[
T \quad \text{snd}_B(\{N_A, N_B, B\}_{PK_A}) : t \in T
\]

Figure 5: Example of the different models: the first transition rule of role \( B \) in NSL.

ones and (a desired subset of) the variables with typed variables. This simple way of enforcing types in the term structure allows us to see that the issue of typing is independent from this work, as it is now not an issue of the protocol model, but rather an issue of the definition of the messages of the protocol.

The second difference is that [24] uses a combination of send and receive events, denoted \( a \rightarrow b : m \), which at first sight looks like a synchronous communication model (where, in particular, the intruder cannot intercept messages), but this is not the case. In recent works, similar events as in our model are used, called Says and Gets [6].

There are further minor differences: in [24], we have two functions \( \text{synth} \) and \( \text{analz} \) that represent each a part of the \( \text{DY} \) closure, namely the synthesis and analysis rules, and instead of \( \text{DY}(IK) \) the closure \( \text{synth}(\text{analz}(IK)) \) is employed, which in general is a proper subset of \( \text{DY}(IK) \) (only when the protocol uses only atomic keys is executed in a typed model, these closures are equal). We thus use the complete \( \text{DY}(IK) \). Also, the way we use the set of permitted instances of agent names is not present in the rules of [24]; this is due to the fact that it does not need to be repeated in each step when given a concrete protocol, but necessary when we have an arbitrary protocol as a message pattern.

4 The Over-Approximation and Persistent Set Rewriting

The intuition about why the models are so similar is that every state fact of the set rewriting model in some way corresponds to the respective send and receive events in the message trace models. We formalize this intuition by a function \( [\cdot] \) that maps a state fact to a set of events:

\[
[\text{state}_A(m_0, \ldots, m_n)] = \{\text{rcv}_A(m_1), \text{snd}_A(m_2), \ldots, \text{rcv}_A(m_{n-1}), \text{snd}_A(m_n)\}
\]

where \( A = \text{player}(m_0) \). Intuitively the other components of \( m_0 \) will later be reflected by the additional condition \( \text{inst}_R(m_0) \) which is not an event. This mapping, together with conditions on the intruder-deducible messages, is the basis for comparing two models, namely to prove inclusions between models and equivalences between models.

We now discuss a phenomenon that is inherent in this message trace model. We observe that in all rules we have only positive conditions on the existence of events on a given trace, with the exception of the creation of fresh data. For instance, the NSL example of Fig. 2 allows the trace shown in Fig. 6 which demonstrates the over-approximation. For brevity, we have omitted the steps with the server \( S \), and also used the \( a \rightarrow b : m \) notation to abbreviate the two events \( \text{snd}_a(m) \) and \( \text{rcv}_b(m) \); moreover, we have labelled these events with the number of the protocol message as which the sender meant it and distinguished “different” sessions using primed labels. In this trace, we have one normal execution of the protocol (steps 1-3). But then in step 2', \( b \) chooses to react a second time to the first message of \( a \). This is because there is nothing in the rule for \( B \) that says that he may not react to a message, to which he has already reacted. Even worse, at this point he has already received an answer to his first reaction, so in fact the session should already be over at this point. However, \( b \) chooses the fresh nonce \( n_3 \) which he links with \( a \)'s first nonce \( n_1 \), so from his point of view there are two different sessions, although \( a \) has only sent one message. Now in step 3' agent \( a \) reacts to \( b \)'s second reply to the initial message from \( a \)—observe
that both step 2, and step 2.’ contain the same nonce $n_1$. This means $a$ does not remember that after its step 3 the nonce $n_1$ is already used and that it should rather not accept messages with this nonce anymore. Finally, $a$ sends in step 3.” another answer to step 2.

More generally, this phenomenon stems from the fact that there is no notion of local state of the agents in the message trace model—all “memory” agents have in this model lies in the messages exchanged. Basically that would be enough to reconstruct each agent’s local state, however, that would require negative predicates on the messages exchanged. For instance, in the case of NSL (without key-server for simplicity), the reaction of role $B$ might look like this:

$$
\begin{align*}
1. & a \rightarrow b: \{n_1, a\}_{pk(b)} \\
2. & b \rightarrow a: \{n_1, n_2, b\}_{pk(a)} \\
3. & a \rightarrow b: \{n_2\}_{pk(b)}
\end{align*}
$$

Figure 6: A trace of NSL demonstrating the over-approximation.

Note that one must use a fresh variable $N'_B$ here to express that $B$ has not sent any answer to $A$’s message with that $N_A$.

In the case of the NSL, this phenomenon does not seem to be a problem—as far as secrecy and certain forms of authentication are concerned. However injective agreement as defined by [21], which additionally checks for replay of messages, is trivially violated for every protocol in this model, since every agent is willing to accept incoming messages of the correct format any number of times. This issue is discussed in Sec. 6. Similarly, many protocols cannot be reasonably modeled with this message trace based model. As an example, the contract signing protocol ASW [3] involves a trusted third party (TTP) which is contacted in case of a dispute. Without going into the details of this protocol, the TTP can give out abort tokens or replacement contracts, and it is essential that the TTP never gives out a replacement contract for a (partial) session that it has already aborted. Thus the steps of the TTP can only be performed if certain events have not yet occurred, and the protocol has trivial false positives otherwise. It should be noted that the formalization of this protocol (see for instance [19]) is beyond the scope of the message sequence specification we have introduced in Sec. 3.

Thus, due to the over-approximation, for a given protocol and goal, we might not be able to prove in the message trace model that the protocol fulfills the goal, even if that is the case in a more precise model. However, we are on the safe side, if we can show that an over-approximated model is indeed safe—as long as this model is indeed an over-approximation of the desired system. Thus, we need to formally prove that the message trace model is indeed an over-approximation of the set rewriting model, i.e. that every state reachable in the set rewriting model of a protocol corresponds to a trace in the message trace model, in a sense to be made precise below.

### 4.1 Persistent Set Rewriting

We now show that there is an over-approximation of the set rewriting model, the persistent set rewriting model, which in turn is an over-approximation of the message trace model. The advantage of taking this little “detour” via persistent rewriting is a precise understanding of the relationships of the models.

Intuitively, the phenomenon just described can be understood as follows: in every transition, an agent forks into two incarnations: one incarnation learns the new incoming message, and creates and remembers the outgoing message of this step. The other incarnation just remains in the state it is, to later possibly spawn further incarnations from exactly this local state. Proving formally the inclusion relationships, we can formalize this intuition via the persistent set rewriting model.
For the persistent set rewriting model, we change the honest agent rules of the set rewriting model as follows: every state fact of the left-hand side is also repeated on the right-hand side, formalizing that one incarnation of the agent remains in its present state. The honest agents rules thus look as follows:

\[
\text{state}^R(m_0^R, \ldots, m_{n-2}^R).\text{iknows}(m_{n-1}^R)
\]

\[
= [\text{fresh}^R] \Rightarrow \text{iknows}(m_0^R).\text{state}^R(m_0^R, \ldots, m_{n-2}^R).\text{state}^R(m_0^R, \ldots, m_n^R)
\]

Note that this is the same construction that we have already implicitly defined for the i\text{knows}-facts. Now all facts contained in a state are also contained in all successor states, hence the name “persistent”. We denote the reachable states of the persistent set rewriting model as \(P\).

### 4.2 Relation to Set Rewriting

We first show that the persistent set rewriting model is indeed an over-approximation of the set rewriting model, in the sense that every reachable state of \(R\) is subsumed by one in \(P\):

**Theorem 1.** For every reachable state \(s \in R\), there is a reachable state \(s' \in P\) such that \(s \subseteq s'\) and \(DY(s) = DY(s')\).

**Proof.** This is shown by structural induction, where we have to slightly strengthen the induction hypothesis in that corresponding states \(s\) and \(s'\) have the same set of used fresh identifiers (i.e. constants from \(\Sigma_0\)).

As the induction basis, observe the initial states are identical in both models. For the induction step, consider a reachable state \(s\) of the set rewriting model, a transition rule \(LHS \Rightarrow RHS\) of the set rewriting model and a substitution \(\sigma\) such that \(LHS\sigma \subseteq s\). (Note that \(\sigma\) also assigns values to the fresh data in \(V\).) The induction hypothesis is that there is a state \(s' \supseteq s\) reachable in the persistent set rewriting model such that \(DY(s) = DY(s')\) and \(s\) and \(s'\) have the same set of fresh constants used.

We show that for the successor state \(t = (s \setminus LHS\sigma) \cup RHS\sigma\) in the set rewriting model, there is a corresponding state \(t' \supseteq t\) reachable in one transition from \(s'\) the persistent set rewriting model. We use the corresponding rule in the persistent set rewriting model, which is \(LHS \Rightarrow RHS\) (we exploit in this notation that the i\text{knows} facts are implicitly persistent). This rule is applicable to \(s'\) with the same substitution \(\sigma\), since \(s' \supseteq s \supseteq LHS\sigma\) and \(s\) and \(s'\) use the same set of fresh constants from \(\Sigma_0\). The resulting new state is

\[
t' = (s' \setminus LHS\sigma) \cup LHS\sigma \cup RHS\sigma = s' \cup RHS\sigma \supseteq s \cup RHS\sigma \supseteq t,
\]

It also follows that \(s'\) and \(t'\) still use the same set of fresh constants. To see that \(t\) and \(t'\) have the same intruder knowledge, observe that \(DY(IK_1 \cup IK_2) = DY(DY(IK_1) \cup IK_2)\), as \(DY\) is a closure operation. Now we have that

\[
DY(t') = DY(s' \cup RHS\sigma) = DY(DY(s') \cup RHS\sigma) = DY(DY(s) \cup RHS\sigma) = DY(s \cup RHS\sigma)
\]

where we have used that i\text{knows}(\cdot) facts are implicitly monotonic and thus \(\{m \mid \text{iknows}(m) \in s\} = \{m \mid \text{iknows}(m) \in (s \setminus LHS\sigma) \cup RHS\sigma\}\). This proves the induction hypothesis.

This implies, in particular, that for every reachable state \(s\) in the set rewriting model, there is a state \(s'\) in the persistent set rewriting model, such that the set of messages the intruder can derive in \(s'\) is at least as large as in \(s\), and that every state fact of \(s\) is contained in \(s'\).
4.3 Inclusion in the Inductive Message Trace Model

The next step is to show that the persistent set rewriting model is subsumed in the message trace model. We do this with respect to the intruder knowledge and the local states of honest agents as formalized by the \([\cdot]\) function. To that end, we extend the definition of \(DY(\cdot)\) to a set of facts \(s\), namely \(DY(s) = DY(\{m \mid iknows(m) \in s\})\), and to a set of events \(E\), namely \(DY(E) = DY(IK_0 \cup \{m \mid snd_a(m) \in \{t\}\})\), given the initial intruder knowledge \(IK_0\) attached to a protocol.

**Theorem 2.** For every reachable state \(s \in \mathcal{P}\), there is a trace \(t \in \mathcal{T}\) such that \(DY(s) = DY([t])\) and for every state fact \(f \in s\), \([f]\subseteq [t]\).

**Proof.** We use again structural induction, in this case on the inductive structure of \(\mathcal{P}\), and again we strengthen the induction hypothesis so that the states \(s\) and the trace \(t\) in question use the same set fresh constants from \(\Sigma_0\).

**Induction Basis.** For the initial state of the persistent set rewriting model and the empty trace \(<\geq \in \mathcal{T}\) fulfills the assumption, as there are no state facts in the initial state (and thus no send and receive events are required on the trace), there are no fresh data and what the intruder can derive is in both cases \(DY(IK_0)\).

**Induction Step.** Given any reachable state \(s \in \mathcal{P}\) and a trace \(t \in \mathcal{T}\) that corresponds to \(s\) in the sense of the induction hypothesis. We show that for every state \(s'\) reachable from \(s\) by one application of a rewrite rule, we can find an extension \(t'\) of \(t\) in \(\mathcal{T}\) that fulfills the induction hypothesis.

Consider any transition rule applicable to \(s\) and the successor state \(s'\), as well as a trace \(t\) that corresponds to \(s\) according to the induction hypothesis; we distinguish three cases, one for each kind of rule:

1. The rule is an initialization rule, i.e. of the form \(\Rightarrow state_R(m)\) such that \(inst_R(m)\) holds. This rule neither creates fresh data nor increases the intruder knowledge. \(m\) is a ground instance of \(m^R_0\), thus \([state_R(m)] = \emptyset \subseteq \{t\}\). Therefore, \(t\) already satisfies the induction hypothesis.

2. The rule is an intruder rule. This adds \(iknows(m)\) where \(m \in DY(s)\), thus \(DY(s') = DY(s) = DY([t])\). Also, no fresh data are created and state facts are created. So again, \(t\) already satisfies the induction hypothesis for \(s'\).

3. The rule is an honest agent state transition rule, i.e. of the form

\[
\text{state}_R(m^R_0, \ldots, m^R_{n-2}).iknows(m^R_{n-1}) = \text{[fresh}_R^n \Rightarrow iknows(m^R_n)).state_R(m^R_0, \ldots, m^R_{n-2}).state_R(m^R_0, \ldots, m^R_n),
\]

for some \(R\) and \(0 < n \leq \text{last}_R\). Let \(\sigma\) be a substitution under which the rule is applied. It follows that \(m^R_{n-1}\sigma \in DY(s) = DY(t)\). Thus, we can apply the intruder rule of the message trace model to obtain the trace \(t' = \text{rcv}_a(m^R_\sigma) : t \in \mathcal{T}\) where \(a = \text{player}(m^R_\sigma)\). Note that \(t\) and \(t'\) have the same intruder knowledge and use the same set of fresh constants. Thus, \(\text{fresh}_R^n\sigma\) are constants that are not yet used in \(t'\). From the construction, it follows that \(\text{inst}_R(m^R_0\sigma)\). By the induction hypothesis, \([\text{state}_R(m^R_0, \ldots, m^R_{n-2})\sigma] = \{\text{rcv}_a(m^R_\sigma), \text{snd}_a(m^R_\sigma), \ldots, \text{snd}_a(m^R_{n-2}\sigma)\} \subseteq \{t\} \subseteq \{t'\}\). This gives us all premises in order to apply the corresponding inductive rule of the message trace model to \(t'\), yielding the trace \(t'' = \text{snd}_a(m^R_n\sigma) : t' \in \mathcal{T}\). \(t''\) uses the same amount of fresh data as \(s'\), moreover for the new state fact of \(s'\) we have

\[
[\text{state}_R(m^R_0, \ldots, m^R_n)\sigma] = [\text{state}_R(m^R_0, \ldots, m^R_{n-2})\sigma] \cup \{\text{rcv}_a(m^R_{n-1}\sigma), \text{snd}_a(m^R_n\sigma)\} \subseteq \{t''\}.
\]

For the intruder knowledge we have

\[
DY(s') = DY(s \cup \{iknows(m^R_n)\sigma\}) = DY([t] \cup \{\text{snd}_a(m^R_n)\sigma\}) = DY(t'').
\]
1. $A \rightarrow B : \{N_A, A\}_{pk(B)}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{pk(A)}$
3. $A \rightarrow B : \{N_A, N'_A\}_{pk(B)}$

Figure 7: A protocol and a “cuckoo egg” trace in the message trace model. On the right of the trace the (simplified) agent-states that would correspond to the trace.

Therefore, $t'' \in \mathbb{T}$ corresponds to $s'$ in the sense of the induction hypothesis.

This completes the induction step.

This shows that $\mathbb{T}$ is an over-approximation of $\mathbb{P}$. We now give an example that the converse direction of this inclusion relationship does not hold in general, i.e. there are traces in the $\mathbb{T}$ model which have no counter part in the $\mathbb{P}$ model.

**Example.** Consider the protocol displayed in Fig. 7, which is similar to NSL, but an important difference is that steps 3. and 4. are independent of what happened in step 2., namely from the nonce $N_B$ created by $B$. The message trace model now admits the trace also shown in Fig. 7. At the first sight this might look like the phenomena we have seen before, but here is something different. Interpreting every line $a \rightarrow b : m$ as the events $\text{snd}_a(m)$ and $\text{rcv}_b(m)$, then we obtain the state facts also shown in Fig. 7 that correspond to the trace with respect to the $\{\} \mathbb{T}$ function. We have simplified the presentation of the state facts so that we only display the involved constants in place of the protocol variables $N_A$, $N_B$, and $N'_A$, respectively. Let us have a closer look at the state fact $\text{state}_{roleA}(n_1, n_4, n_3)$ which is created by step 2.' of the trace. This follows by interpreting steps 1., 2., and 3. as “one session”. Observe that there is already $\text{state}_{A}(n_1, n_2, n_3)$ which is created by step 2. of the trace. This follows by interpreting steps 1., 2., and 3. as one session. It should now become obvious why we call this a cuckoo egg trace: After a complete run of the protocol, a new message can come in that could be interpreted as belonging to the run; now an agent can breed-out this cuckoo egg in the sense that he replaces an old message in his local state with the cuckoo-egg. In the example, the cuckoo-egg $n_4$ replaces the original $n_2$ in the local state of $A$.

In the set rewriting and the persistent set rewriting model, there cannot be a counter part for such a trace. The reason is that simulating the run would create fresh nonces for $N'_A$, namely once we have $\text{state}_{A}(n_1, n_2, n_3)$, we cannot obtain the state $\text{state}_{A}(n_1, n_4, n_3)$ anymore, but the last nonce $n_3$ is necessarily some other fresh constant in the (persistent) set rewriting model. The cuckoo-egg $n_4$ is thus the reason why certain traces of the message trace model have no counter part in the (persistent) set rewriting model, and thus, we cannot show equivalence of the two models. We will see below, however, that for protocols with data abstraction, we can indeed prove the equivalence of the models. In other words, persistent set rewriting can breed-out all cuckoo eggs provided that no eggs are fresh.

## 5 Reachable Facts and Events

In the previous section we have shown that a large part of the control structure of protocols is abstracted away in the $\mathbb{P}$ and $\mathbb{T}$ models, namely the order in which events have occurred does not play any role and the set of reached local states of honest agents grows monotonically. An obvious idea might thus be to simplify matters further by considering the set of reachable facts $\mathbb{F} = \bigcup_{s \in \mathbb{P}} s$ and the set of reachable events $\mathbb{E} = \bigcup_{t \in \mathbb{T}} [t]$. This drastically simplifies many search problems, as
instead of searching an infinite state search space, we have to consider only the facts that occur in any state.\footnote{The set of reachable facts is finite under certain conditions, namely when data abstraction is performed, as discussed below, and when the intruder is restricted to well-typed messages.}

However, there are two problems related to this. Firstly, authentication goals cannot be formalized in the usual way anymore as discussed in Sec. 6. Secondly, the fresh data in every state or trace is chosen based on what has not yet been used. Collecting facts and events from different traces would give a kind of collision, e.g. the fresh data in a session between an honest agent and the intruder (which the intruder may know) and the fresh data in a session between honest agents (which the intruder may not know) can be identical. This renders the F and E models useless for protocols with fresh data.

Another view on this is the following: both \( \mathbb{P} \) and \( T \) are inductively defined. Usually, all rules of an inductive definition are monotonic, i.e. if a rule is applicable to a set \( S \), then it is also applicable to any superset of \( S \). This guarantees that the fixed-point, i.e. the least set closed under the rules, is uniquely defined. If one instead tries an inductive definition of the set of reachable facts and events, then there is the problem of the negative condition that fresh data are constants not used so far. Leaving out this condition renders the model useless (as every protocol will then have trivial attacks). Including the condition, however, leads to non-monotonic induction rules and the fixed-point is in this case not uniquely defined, but rather depends on the order in which the inductive rules have been applied.

However, in many automated verification approaches a kind of data abstraction is performed [7, 9, 10, 11, 13, 18, 25]—inspired by the idea of abstract interpretation [15]. The idea is the following: if we map the infinite set of (fresh) data to finitely many equivalence classes and instead of each concrete datum rather consider its abstract equivalence class, many search problems become finite and the abstract model is an over-approximation of the concrete one. This argumentation has, however, a prerequisite: there may not be any negative comparisons in the rules, e.g. that two nonces are not the same, or else the abstract model would not be an over-approximation of the concrete one.

We do not discuss the matters of data abstraction here, but we consider protocol models where no fresh data occur, and show that under these circumstances we can safely perform the simplification to reachable facts and events models, and moreover, that the reachable facts and the reachable events models are then equivalent in the sense we have defined before. As an example how an abstract model of NSL can look like, consider the following data abstraction: the nonces are replaced by \( \text{na}(A, B) \) in the rules for \( A \) and the fresh nonces created by role \( B \) are replaced by \( \text{nb}(B, A) \) in the rules for \( B \), where \( \text{na} \) and \( \text{nb} \) are new binary function symbols. As an intuition, one may think of agents who always use the same nonce when talking to the same agents. Intuitively it should be clear that, if the protocol is safe under such a behavior of the agents, then it is also safe in case the agents indeed freshly create the nonces in each session.

We say that a protocol is \textit{data abstract} (or \textit{has no fresh data}) iff \( \text{fresh}^R_n = \emptyset \) for all roles \( R \) and all even indices \( 0 < n \leq \text{last}^R \). For such a protocol, we can now show the converse direction of the inclusion relation of Theorem 2, namely that every trace of the message trace model has a counter part (in terms of intruder knowledge and local agent states) in the persistent set rewriting model:

**Theorem 3.** If the protocol is data abstract, then for all \( t \in T \) there exists a state \( s \in \mathbb{P} \) such that \( \mathcal{DY}([t]) = \mathcal{DY}(s) \) and all agent states represented by \( t \) are contained in \( s \): \( \{ \text{state}_R(m_0, \ldots, m_n) | [\text{state}_R(m_0, \ldots, m_n)] \subseteq [t] \wedge \text{inst}_R(m_0) \} \subseteq s \).

**Proof.** This proof uses the structure of the inductive definition of \( T \).

**Induction Basis.** Consider the empty trace \( <> \in T \). Its intruder knowledge is \( \mathcal{DY}(IK_0) \), it has no fresh data. Moreover, since there are no send and receive events in the empty trace, the following holds:

\[
\{ \text{state}_R(m_0, \ldots, m_n) | [\text{state}_R(m_0, \ldots, m_n)] \subseteq [t] \wedge \text{inst}_R(m_0) \} = \{ \text{state}_R(m_0) | \text{inst}_R(m_0) \}
\]
Here it becomes important that the set of all \( m \), for which \( \text{inst}_R(m) \) holds, is finite, otherwise we could not reach—in finitely many steps—a state of the \( \mathbb{P} \) model that corresponds to the empty trace. (And a similar problem would arise in the induction step below.) For finitely many such \( m \), however, a straightforward induction proof shows that we can reach a state \( s \) from the initial state \( s_0 \) of \( \mathbb{P} \), which contains all required state facts, using the initialization rules of the \( \mathbb{P} \) model. It holds that \( \mathcal{DY}(s) = \mathcal{DY}(s_0) = \mathcal{DY}(IK_0) \), and there are no fresh data used in \( s \). Therefore \( s \) corresponds to the empty trace in the sense of the induction hypothesis.

Induction Step. Given any trace \( t \in \mathcal{T} \), for which we already have a corresponding state \( s \in \mathbb{P} \) in the sense of the induction hypothesis. We now show that for every extension \( t' \) of \( t \) that we can obtain by using a rule of \( \mathcal{T} \), we can find a state \( s' \in \mathbb{P} \) that can be reached from \( s \) in finitely many steps of the \( \mathbb{P} \) model and that also satisfies the induction hypothesis with respect to \( t' \).

Let \( \text{states}(t) = \{ \text{state}_R(m_0, \ldots, m_n) \mid [\text{state}_R(m_0, \ldots, m_n)] \subseteq [t] \wedge \text{inst}_R(m) \} \) denote the set of state facts associated with a trace \( t \). First observe that, even though \( t' \) is an extension of \( t \) with just one event, \( \text{states}(t') \setminus \text{states}(t) \) can contain more than one state fact. In other words, a single event in the \( \mathcal{T} \) model may correspond to multiple steps of the \( \mathbb{P} \) model, as demonstrated by the example above.

We first show that \( \text{states}(t) \) is finite for every trace \( t \in \mathcal{T} \) (otherwise we would be unable to reach a corresponding state of \( \mathbb{P} \) in finitely many steps). This can be shown by a simple inductive proof: the empty trace has a finite corresponding set of states as seen above, and with every extension \( t' \) of a trace \( t \) by one event, the set remains also finite, as in the “worst case” this new event can induce a finite number of new state facts for every state fact of \( \text{states}(t) \).

Next, we remark a monotonicity property of \([t]\) in the way a state fact “grows”, namely \([\text{state}_R(m_0, \ldots, m_n)] \supseteq [\text{state}_R(m_0, \ldots, m_{n-2})] \) for all even \( 0 < n \leq \text{last} \). This induces a well-founded partial order on \( \text{states}(t') \), namely the order \([t]\) \supseteq \([t']\) which cannot have an infinite descending chain (since \([t']\) is finite) and, moreover, every descending chain is eventually founded in \([t]\). Let \( f_1, \ldots, f_n \) be a linearization of the facts of \( \text{states}(t') \setminus \text{states}(t) \) in this partial order, i.e. each \( f_i \) is an extension of a state fact in \( \text{states}(t) \cup \{ f_1, \ldots, f_{i-1} \} \).

We now show by an “inner” induction that we can construct a sequence \( s = s_0 \Rightarrow s_1 \Rightarrow \cdots \Rightarrow s_n = s' \) of reachable states of \( \mathbb{P} \), such that for some sets of messages \( \text{IK}_i \subseteq \mathcal{DY}(t'[i]) \) with \( \text{IK}_{i+1} \supseteq \text{IK}_i \) it holds that \( s_i = s \cup \{ f_1, \ldots, f_i \} \cup \{ \text{iknows}(m) \mid m \in \text{IK}_i \} \).

Inner Induction Basis: for \( s_0 \) (and \( \text{IK}_0 = \emptyset \)) nothing is to show.

Inner Induction Step: Suppose we have already shown reachability of \( s_i \) with the above conditions and a respective \( \text{IK}_i \). We show we can then reach \( s_{i+1} \) for some \( \text{IK}_{i+1} \supseteq \text{IK}_i \). The state fact \( f_{i+1} \), for which we have to show a way to reach it, must have the form \( \text{state}_R(m_0^R, \ldots, m_{n-2}^R) \sigma \) for some substitution \( \sigma \) (due to the definition of \([t]\)) and some even \( 0 < n \leq \text{last} \). Also, by the inner induction hypothesis, \( \text{state}_R(m_0^R, \ldots, m_{n-2}^R) \sigma \) is already contained in \( s_i \). We thus have to show that the intruder can construct the message \( m_{n-1}^R \sigma \), so we can apply the respective transition rule of \( \mathbb{P} \). The respective receive event, namely \( \text{rcv}_a(m_{n-1}^R \sigma) \) for a = \text{player}(m_{n-1}^R \sigma) must be present in \( t \) already (otherwise the extension to \( t' \) is not possible). Therefore \( m_{n-1}^R \sigma \in \mathcal{DY}([t]) = \mathcal{DY}(s) \), by the outer induction hypothesis. Thus, we can obtain the fact \( \text{iknows}(m_{n-1}^R \sigma) \) by finitely many applications of intruder deduction rules on \( s_i \), i.e. there is a state \( s_i' \) with \( s_i \rightarrow s_i' \in \mathbb{P} \) and \( s_i' = s_i \cup \{ \text{iknows}(m) \mid m \in \text{IK}_i \} \) for some set \( \text{IK}_i \subseteq \text{IK}' \subseteq \mathcal{DY}(t'[i]) \). We can now finally apply the transition rule to \( s_i' \) which gives use \( s_{i+1} = s_i' \cup \{ \text{state}_R(m_0^R, \ldots, m_{n-2}^R) \sigma, \text{iknows}(m_{n-1}^R \sigma) \} \). Define \( \text{IK}_{i+1} = \text{IK}' \cup \{ m_{n-1}^R \sigma \} \). Then \( s_{i+1} \) and \( \text{IK}_{i+1} \) satisfy the inner induction hypothesis: \( \text{IK}_{i+1} \subseteq \mathcal{DY}([t']) \), \( s_{i+1} \) is reachable in finitely many steps from \( s_i \) and \( s_{i+1} = s_i \cup \{ f_{i+1} \} \cup \{ \text{iknows}(m) \} \mid m \in \text{IK}_{i+1} \).

From the inner induction proof, we can now show that the outer induction hypothesis holds for \( t' \) and \( s' \): \( \mathcal{DY}(s') = \mathcal{DY}(s \cup \text{IK}_n) = \mathcal{DY}([t']) \) and \( \text{states}(t') \subseteq s' \).

By the combination of Theorem 2 and 3, the persistent set rewriting model and the inductive trace model thus coincide for protocols under data abstraction.

We now show that it is safe to turn from the persistent set rewriting model to the reachable facts model. The over-approximation relationship directly follows from the definition, in the sense that every fact of a reachable state is present in \( \mathbb{F} \). What we prove here is that the reachable facts
model does not “merge too many facts”, in the sense that every subset of the reachable facts that is contained in some reachable state of the persistent set rewriting model. The following Theorem states this and an analogous statement for the reachable events and message traces models:

**Theorem 4.** If the protocol has no fresh data, then for every finite set of reachable facts $F \subseteq \mathcal{F}$, there is a reachable state $S \in \mathcal{P}$ such that $F \subseteq S$. Similarly, for every finite set of reachable events $E \subseteq \mathcal{E}$, there is a trace $t \in \mathcal{T}$ such that $E \subseteq [t]$.

**Proof.** Reachable states and facts. Given a finite set $F = \{f_1, \ldots, f_n\} \subseteq \mathcal{F}$, then by definition of $\mathcal{F}$, for each $f_i$ there is a state $S_i \in \mathcal{P}$, such that $f_i \in S_i$. Thus it is sufficient to show that for any reachable states $S_1, S_2 \in \mathcal{P}$, their union $S_1 \cup S_2 \in \mathcal{P}$ is reachable. Then the proposition follows by inductively applying this argument to all $S_i$ for each $f_i$. Let $S_1, S_2 \in \mathcal{P}$, and let $S_0 \in \mathcal{P}$ be the initial state of the persistent set rewriting model. Then $S_0 \subseteq S_1$ and $S_0 \subseteq S_2$, as the rewriting is persistent. Since no fresh data are created, the same rule under the same substitution can be applied to $S_1$ as to $S_0$, so to obtain the facts of $S_2$. Thus $S_1 \cup S_2 \in \mathcal{P}$.

Reachable traces and events. The argument is completely analogous with $[\cdot]$ applied to the traces. □

Note that from this proof we can see that the reachable facts and events fixed-point are only meaningful because $DY(IK)$ is itself monotone in $IK$ (i.e. the more messages the intruder has seen, the more he can deduce).

We conclude this section with a consequence of all previous Theorems, namely that for protocols under data abstraction, also the reachable facts and reachable events models coincide:

**Theorem 5.** Consider a data abstract protocol. Then $DY(\mathcal{F}) = DY(\mathcal{E})$, and moreover it holds that $\text{state}_R(m_0, \ldots, m_n) \in \mathcal{F}$ iff $\text{inst}_R(m_0)$ and $[\text{state}_R(m_0, \ldots, m_n)] \subseteq \mathcal{E}$.

**Proof.** $DY(\mathcal{F}) \overset{\text{Def}}{=} DY(\cup_{s \in \mathcal{S}})$ Th.2 & Th.3 $DY(\cup_{t \in \mathcal{T}}[t]) \overset{\text{Def}}{=} E \overset{\text{Def}}{=} DY(\mathcal{E})$.

Let $\text{state}_R(m_0, \ldots, m_n) \in \mathcal{F}$. Then $\text{inst}_R(m_0)$, and there is $s \in \mathcal{P}$ with $\text{state}_R(m_0, \ldots, m_n) \in s$. By Theorem 2, there is a trace $t \in \mathcal{T}$, such that $[\text{state}_R(m_0, \ldots, m_n)] \subseteq [t]$. Thus

$$[\text{state}_R(m_0, \ldots, m_n)] \subseteq \mathcal{E}.$$  

Let $[\text{state}_R(m_0, \ldots, m_n)] \subseteq \mathcal{E}$ and $\text{inst}_R(m_0)$. Then by Theorem 4, there is a trace $t \in \mathcal{T}$ such that $[\text{state}_R(m_0, \ldots, m_n)] \subseteq [t]$. By Theorem 3, there is a reachable state $s \in \mathcal{P}$, such that $\text{state}_R(m_0, \ldots, m_n) \in s$. Thus $\text{state}_R(m_0, \ldots, m_n) \in \mathcal{P}$. □

This concludes the argumentation on the relation of the persistent rewriting model and the message trace model: if we have no fresh data in messages, we can move to a reachable facts and reachable events model, respectively, without any loss of generality, and then they are equivalent.

Several abstract verification approaches, e.g. [9, 13, 18], go yet a step further in the approximation and consider only message terms, i.e. ignoring the information whether messages were sent or received. The idea is to not even distinguish between the messages the intruder sends and the messages an honest agent sends, thus we focus on a pure intruder deduction problem. We define the model $\mathcal{M}$ as the least set of messages that contains $IK_0$ and that is closed under the rules shown in Fig. 8 and the rules of $DY$. With this model, we have finally arrived at a stage where the “entire protocol is abstracted away” resulting in a pure intruder deduction problem: in fact we see the behavior of honest agents just as additional intruder deduction rules. In fact the honest agents are now nothing more than oracles in the terminology of [13] for the intruder: the
intruder can ask them questions of a certain form (the messages in the premises of a rule), and they will give a reply as a function of the question (the consequence of the rule). The model $M$ is again an over-approximation of the previous models:

**Theorem 6.** $M \supseteq DY(E)$.

*Proof.* Consider a total order on the events in $E = \{e_1, e_2, \ldots\}$ such that $e_i$ is the consequence of a rule of the $E$ model for some subset of $\{e_1, \ldots, e_{i-1}\}$ as premises. (This is possible due to the inductive definition of $E$.) We show $DY(\{e_1, \ldots, e_i\}) \subseteq M$ by induction over $i$.

For the base case $i = 0$ we have $DY(\{< >\}) \subseteq M$.

For the induction step, assume $DY(\{e_1, \ldots, e_i\}) \subseteq M$. It is to show that this also holds for $i + 1$. We distinguish the cases that $e_{i+1}$ is a send or receive event.

If $e_{i+1} = rcv_a(m)$, then $m \in DY(\{e_1, \ldots, e_i\}) \subseteq M$, by the induction hypothesis. Since $M$ is closed under the rules of $DY$, also $m \in M$.

If $e_{i+1} = snd_a(m)$, then $rcv_a(m_0^R)\sigma, \ldots, rcv_a(m_{n-1}^R)\sigma \in F$ for some $R$, even index $0 < n \leq last^R$ and substitution $\sigma$ such that $a = player(m_i^R)\sigma$ and $m = m_n^R\sigma$ and $inst_R(m_0\sigma)$. By the induction hypothesis we therefore have $m_1^R\sigma, \ldots, m_{n-1}^R\sigma \in M$, thus by the rules of $M$, also $m_n^R\sigma \in M$. \hfill $\Box$

### 6 Goals

We have so far ignored the question of how to describe the properties that a protocol is supposed to ensure, or conversely, what states or traces would count as attacks to the protocol. Based on the results from the previous sections, namely the correspondence between local states of honest agents on the one side, and receive and send events on the other side, we now investigate how one can formalize standard goals in the different models.

#### 6.1 Secrecy

We first consider secrecy goals which are straight-forward to handle: we can apply Theorems 1, 2, 4, and 5 to show that, for instance, if the intruder can find out a secret in a reachable state of $R$, then there is a trace of $T$ in which the intruder has also found out that secret.

For the message pattern approach specification, a we can specify a secrecy goal by means of a role $R$, a subterm of a message pattern of $R$ and a set of protocol variables (the latter representing the people who may know it). For instance, in the example of NSL, we could specify that the protocol variables that for $R$, both the subterms $N_A$ and $N_B$ of $A$ are secrets shared between the people instantiating the protocol variables $A$ and $B$ of $A$. (And one can specify similar goals from $B$’s point of view.)

In the set rewriting model, a violation of secrecy is thus defined as a state in which the intruder knows a message that is supposed to be secret between a set of agents that he does not belong to. More concretely, for the NSL example, an attack state for secrecy of $N_A$ and $N_B$ from $A$’s point of view is thus formulated as follows:

\[
\exists s \in R. \exists \sigma[dom = vars(A)].
\]

\[
\begin{align*}
\text{state}_A(A, B, \ldots, \{N_A, A\}_{PK_A}, \ldots, \{N_A, N_B, A\}_{PK(A)}, \{N_B\}_{PK_B})\sigma \in s \land \\
A\sigma \neq i \land B\sigma \neq i \land (iknows(N_A\sigma) \in s \lor iknows(N_B\sigma) \in s)
\end{align*}
\]

Here, we have existentially quantified over a substitution of all protocol variables of $A$ (where $\sigma[dom = V]$ expresses the restriction that $\sigma$ is a substitution with domain $V$). Note that for an
attack, also the attack trace is of interest, i.e. the set of messages matched for the im knows(·) facts in the rule applications that lead from the initial state to the attack state.

More generally, for each of the different models, we now define a predicate that defines, what a violation of a secrecy goal means. The particular secrecy goal is specified by a triple \((R, M, \{A_1, \ldots, A_n\})\) where \(R\) is a role, \(M\) is a subterm of one of \(R\)'s message patterns (i.e. \(m_{R}^{0}, \ldots, m_{last R}^{R}\)), and each \(A_i\) is a variable that appears in these message patterns. Intuitively, it means that every message that instantiates \(M\) in a concrete run of the protocol of \(R\) must be kept secret between the agents instantiating the variables \(\{A_1, \ldots, A_n\}\); in particular, the intruder may not find out the instantiation of \(M\) unless he is one of the \(A_i\). The formal definition for the different models is as follows:

\[
\text{secrecyFlaw}_R(R, M, \{A_1, \ldots, A_n\}) = \exists s \in R. \exists \sigma.
\]

\[
\text{state}_R(m_0^R, \ldots, m_{last R}^R \sigma) \in s \land i \notin \{A_1 \sigma, \ldots, A_n \sigma\} \land \text{im knows}(M \sigma) \in s
\]

\[
\text{secrecyFlaw}_P(R, M, \{A_1, \ldots, A_n\}) = \exists s \in P. \exists \sigma.
\]

\[
\text{state}_R(m_0^R, \ldots, m_{last R}^R \sigma) \in s \land i \notin \{A_1 \sigma, \ldots, A_n \sigma\} \land \text{im knows}(M \sigma) \in s
\]

\[
\text{secrecyFlaw}_E(R, M, \{A_1, \ldots, A_n\}) = \exists s \in E. \exists \sigma.
\]

\[
[ \text{state}_R(m_0^R, \ldots, m_{last R}^R \sigma) ] \subseteq [it] \land i \notin \{A_1 \sigma, \ldots, A_n \sigma\} \land M \sigma \in DY([it])
\]

\[
\text{secrecyFlaw}_S(R, M, \{A_1, \ldots, A_n\}) = \exists \sigma.
\]

\[
\text{state}_R(m_0^R, \ldots, m_{last R}^R \sigma) \in F \land i \notin \{A_1 \sigma, \ldots, A_n \sigma\} \land \text{im knows}(M \sigma) \in F
\]

\[
\text{secrecyFlaw}_S(R, M, \{A_1, \ldots, A_n\}) = \exists \sigma.
\]

\[
[ \text{state}_R(m_0^R, \ldots, m_{last R}^R \sigma) ] \subseteq [E] \land i \notin \{A_1 \sigma, \ldots, A_n \sigma\} \land M \sigma \in DY([E])
\]

\[
\text{secrecyFlaw}_S(R, M, \{A_1, \ldots, A_n\}) = \exists \sigma.
\]

\[
\{m_0^R, \ldots, m_{last R}^R \sigma\} \subseteq M \land i \notin \{A_1 \sigma, \ldots, A_n \sigma\} \land M \sigma \in M
\]

Note that in this definition we use the relationships between the different models that we have established so far, in particular, we use the \([\cdot]\) function to "translate" the agent-state facts into the models that talks about events instead.

The following Theorem tells us that the over-approximations are safe in the sense that if the more precise models have an attack to secrecy, then also the approximated models have an attack to secrecy:

**Theorem 7.** For any secrecy goal \(G\), the following implications hold:

- \(\text{secrecyFlaw}_R(G)\) implies \(\text{secrecyFlaw}_P(G)\),
- \(\text{secrecyFlaw}_P(G)\) implies \(\text{secrecyFlaw}_E(G)\) and \(\text{secrecyFlaw}_P(G)\),
- \(\text{secrecyFlaw}_E(G)\) implies \(\text{secrecyFlaw}_S(G)\),
- \(\text{secrecyFlaw}_S(G)\) implies \(\text{secrecyFlaw}_S(G)\).

**Proof.** The proof is immediate from the previous Theorems 1-6, and we just give the first implication as an example. Suppose \(\text{secrecyFlaw}_R(G)\) holds; then there is a state \(s \in R\) and a substitution \(\sigma\) such that the state predicate that corresponds to the final state of \(R\) under \(\sigma\), none of the variables \(A_i\) is substituted by intruder, and \(M \sigma\) is known to the intruder. By Theorem 1, we thus have that this state fact and this im knows fact are present in some reachable state \(s' \in P\) of the persistent set rewriting model, and thus \(P\) also has this attack.

\(\square\)
Using Data Abstraction for Secrecy Specifications. As explained previously, the models E, F, and M only make sense in the context of data abstraction. This data abstraction is sometimes used for a simpler formulation of the secrecy goals. Let us consider the example of the NSL protocol under the data abstraction, i.e., that for \( N \) used for a simpler formulation of the secrecy goals. Let us consider the example of the NSL protocol.

As explained previously, the models A, B, and S of an agent in \( A \) has the form \( na(A, B) \) due to the data abstraction. Thus every violation of secrecy of \( N \) from \( A \)'s point of view is described by the underivability of the secret \( na(A, B) \) in case \( i \) is neither A nor B.

We note that in a similar way, we can specify the secrecy of \( N \) from \( B \)'s point of view or the secrecy of \( N \) from \( A \)'s point of view. That is because the messages that agents receive in these positions are not necessarily of the form \( na(A, B) \) and \( nb(B, A) \), respectively.

6.2 Authentication (First Attempt)

We will first discuss a standard way to express authentication goals and show that this causes a serious problem with the over-approximation. As an example for authentication, first consider the NSL protocol.

We will first discuss a standard way to express authentication goals and show that this causes a serious problem with the over-approximation. As an example for authentication, first consider the NSL protocol. We discuss the relationship with other formalizations of authentication below, but let us first consider again the NSL protocol.

For the persistent set rewriting model \( P \), we can formulate that \( A \) fails to authenticate \( B \) on nonce \( N_B \) as follows (where \( \sigma[\text{dom} = V] \) expresses the restriction that \( \sigma \) is a substitution with domain \( V \)):

\[
\exists s \in P. \exists \tau[\text{dom} = \text{vars}(A)]. \forall \sigma[\text{dom} = \text{vars}(B) \setminus \text{vars}(A)]. A \sigma \neq i \land B \sigma \neq i \land \\
\text{state}_A(A, B, S, \ldots, \{N_A, A\}_{PK_B}, \{N_A, N_B, B\}_{pk(A)}, \{N_B\}_{PK_B}) \sigma \in s \land \\
\text{state}_B(B, S, \{N_A, A\}_{pk(B)}, \ldots, \{N_A, N_B, B\}_{PK_A}) \sigma \tau \notin s
\]

This formalizes that there is a reachable state in the \( P \) model, such that for certain values of the protocol variables, an agent \( A \) in \( A \) has finished the protocol while in that state no agent \( B \) in \( B \) exists who agrees with \( A \) on their names and the exchanged nonces. Note that we do not demand here that \( B \) shall have finished his run of the protocol, he must have only progressed far enough to have created a nonce \( N_B \). (For the standard set rewriting model, one has to additionally require that no extension of \( B \)'s state fact are in \( s \), or else the goal would lead to false positives.) Also, the formula requires \( A \neq i \) and \( B \neq i \), as sessions where the intruder is a participant have no security guarantees.

We discuss the relationship with other formalizations of authentication below, but let us first consider how such a goal would be encoded in the message trace model T. Using the [\( \cdot \)\] function, we may attempt to formulate the above authentication goal for \( T \) as follows:

\[
\exists t \in T. \exists A, B, S, \ldots, N_A, N_B, PK_A, PK_B \in T_S. A \neq i \land B \neq i \land \\
[\text{state}_A(A, B, S, \ldots, \{N_A, A\}_{PK_B}, \{N_A, N_B, B\}_{PK_A}, \{N_B\}_{PK_B})] \subseteq [t] \land \\
[\text{state}_B(B, S, \{N_A, A\}_{PK_B}, \ldots, \{N_A, N_B, B\}_{PK_A})] \not\subseteq [t]
\]

This is in fact the standard way to express authentication for the message trace model: by the [\( \cdot \)\] function, all state facts get translated into messages sent or received by some agent. The important question is now whether or not this formulation of authentication goals for \( T \) is sound. In other words, does verifying \( T \) for an authentication goal imply verifying \( R \) for that goal?

Before we look at this question, let us give a general definition of authentication flaws for the \( R \), \( P \), and \( T \) model, independent from the concrete protocol example.

An authentication goal is a four-tuple \( (A, B, A, B) \), pronounced as “\( A \) authenticates \( B \)”. The components are as follows: \( A \) and \( B \) are roles, and \( A \) and \( B \) are variables for the names of the
players of \(A\) and \(B\). Note that we do not specify any message that is to be authenticated; rather, we assume that the agents should agree on all protocol variables that appear in both message pattern specifications, i.e. which appear in \(\text{vars}(A) \cap \text{vars}(B)\), where \(\text{vars}(R) = \bigcup_{i = 0}^{\text{last}} \text{vars}(m_i^R)\). In the following, let \(n = \text{last}^A\) and \(l(M) = \min\{i \in \{1, \ldots, \text{last}^B\} \mid M\) subterm of \(m_i^B\) denote the final step in the protocol execution of \(A\) (i.e. where \(A\) accepts \(M\)) as well as the first step in the protocol execution of \(B\) where \(B\) knows \(M\):

\[
\text{authFlaw}_{\mathcal{R}}(A, A, B, B, M) = \\
\exists s \in \mathbb{P}. \exists \bar{\sigma}(\text{dom} = \text{vars}(A)) \\
A \bar{\sigma} \neq i \land B \bar{\sigma} \neq i \land \\
\text{state}_A(m_0^A, \ldots, m_n^A) \bar{\sigma} \in \mathcal{S} \land \\
\text{state}_B(m_0^B, \ldots, m_{l(M)}^B) \bar{\sigma} \notin s \land \ldots \land \text{state}_B(m_0^B, \ldots, m_{l(M)}^B) \bar{\sigma} \notin s \\
\text{authFlaw}_{\mathcal{P}}(A, A, B, B, M) = \\
\exists s \in \mathbb{P}. \exists \bar{\sigma}(\text{dom} = \text{vars}(A)) \\
A \bar{\sigma} \neq i \land B \bar{\sigma} \neq i \land \\
\left[\text{state}_A(m_0^A, \ldots, m_n^A) \bar{\sigma}\right] \subseteq [t] \land \left[\text{state}_B(m_0^B, \ldots, m_{l(M)}^B) \bar{\sigma}\right] \not\subseteq [t] \\
\text{authFlaw}_{\mathcal{S}}(G) = \\
\exists \sigma \in \text{vars}(\mathcal{S}) \exists \tau \exists \bar{\sigma}(\text{dom} = \text{vars}(A)) \\
A \bar{\sigma} \neq i \land B \bar{\sigma} \neq i \land \\
\left[\text{state}_A(m_0^A, \ldots, m_n^A) \bar{\sigma}\right] \subseteq [t] \land \left[\text{state}_B(m_0^B, \ldots, m_{l(M)}^B) \bar{\sigma}\right] \not\subseteq [t]
\]

For the original set rewriting model, we need to enumerate all states of \(B\) from where he first learns \(M\) to his final step, since \(B\) may have progressed in the protocol execution already, while for the persistent set rewriting model, it is sufficient to see whether \(B\) was once in the state where he learned \(M\). For the trace model, we use again the \([\cdot]\) function to translate the goal into events. The situation is not as simple as for the secrecy goals: we cannot now show directly from the Theorems of the previous sections that these goals imply each other. The reason is that the formulation of the authentication goals has also negative conditions about the containment of facts, while the Theorems of the previous sections tell us that the models are in an over-approximation relation, i.e. one model contains at least as much facts/events as the other.

First, we can see that the over-approximation of the persistent set rewriting model is fine:

**Theorem 8.** For any authentication goal \(G\), it holds that \(\text{authFlaw}_{\mathcal{R}}(G)\) implies \(\text{authFlaw}_{\mathcal{P}}(G)\).

**Proof.** Re-inspecting the proof of Theorem 1, we make a stronger statement on the relationship of the set rewriting and the persistent set rewriting models: For every reachable state \(s \in \mathcal{R}\), there is a state \(s' \in \mathcal{P}\), such that \(s' = s \cup \{\text{state}\_\mathcal{R}(m_0^\mathcal{R}, \ldots, m_{n-2}^\mathcal{R}) \mid \text{state}\_\mathcal{R}(m_0^\mathcal{R}, \ldots, m_n^\mathcal{R}) \bar{\sigma} \in s \land n \geq 2\}\), in other words \(s'\) is the least closure of \(s\) under prefixes of local agent states. We first prove this statement by induction over the structure of \(\mathcal{R}\).

For the induction basis, the initial state is identical in \(\mathcal{R}\) and \(\mathcal{P}\). As it contains no state facts it is trivially prefix-closed for the agent states.

For the induction step, let \(s \in \mathcal{R}\) be a reachable state, and \(s' \in \mathcal{P}\) be the prefix-closure of \(s\). To show is that every extension \(t \in \mathcal{R}\) which is reachable in one transition corresponds to a similar transition from \(s'\) to \(t' \in \mathcal{P}\) which is the prefix-closure of \(t'\). If the rule applied in the transition is one of the initialization rules, then the same rule can be applied to \(t'\) adding the same new initial state fact. Since this state fact contains only the first message of the approach, it is also prefix-closed. For the case of an intruder deduction rule, the situation is the same: the same rule can be applied to \(s'\) introducing the same new facts, which cannot introduce further facts according to the prefix-closure. For the case of a transition rule of an honest agent, there may be a new state fact introduced in \(t\) and \(t'\) of the form \(\text{state}\_\mathcal{R}(m_0^\mathcal{R}, \ldots, m_{n-2}^\mathcal{R}) \bar{\sigma} \in s\). By the induction hypothesis, \(f \in s'\) and thus \(s'\) is also prefix closed for \(f\). As \(\mathcal{P}\) is persistent, \(f \in t'\), and also the other state facts from \(t\), thus \(t'\) is prefix-closed for \(\text{state}\_\mathcal{R}(m_0^\mathcal{R}, \ldots, m_n^\mathcal{R}) \bar{\sigma}\). Since no other state facts are added and \(t\) is already prefix-closed for them, \(t'\) is prefix-closed for all state facts.
Finally, the over-approximation between the models can be shown as follows. For every state \( s \in \mathbb{R} \) that violates an authentication goal \( G \), there is a state \( s' \in \mathbb{R} \) which is the prefix-closure of \( s \) by the previous induction proof. Suppose \( s' \) is not an attack state. Then there is a \( \sigma[\text{dom} = \text{vars}(A)] \) such that for every \( \tau[\text{dom} = \text{vars}(B) \setminus \text{vars}(A)] \) we have \( \text{state}_A(m^A_0, \ldots, m^A_{\text{last}A} \sigma) \in s \), while \( \text{state}_B(m_B^0, \ldots, m_B^B \sigma) \not\in s \) for any even \( i \in \{1, \text{last}B\} \). Since \( s' \) is not an attack state, for every \( \tau[\ldots] \), we have that \( \text{state}_B(m_B^0 \text{role}_B, \ldots, m_B^B \sigma) \not\in t \). This contradicts the previous conclusion that there is no such fact in \( t \) for any \( i \in \{1, \ldots, \text{last}B\} \) and any \( \tau[\ldots] \). Therefore, if \( s \) is an attack state, \( s' \) is an attack state.

\[ \square \]

### 6.3 A Subtle Problem About Authentication

We will now show that the first formulation of authentication for \( T \) and \( R \) are not in general equivalent. This is due to the negation in the predicate for the local state of \( B \). Given an attack state of \( R \), we cannot conclude that the “corresponding” trace in \( T \) is still an attack trace: as \( T \) is an over-approximation of \( R \), it may happen that the negative condition (that \( B \) is not in a certain state) does no longer hold in \( T \). Thus, more generally, we cannot be sure that verifying authentication in the \( T \) model indeed implies authentication in the \( R \) model.

It is relatively straight-forward to come up with an example of a protocol and an attack state \( s \in \mathbb{R} \) (for authentication) such that the corresponding trace \( t \in \mathbb{T} \) is not a violation of the authentication goal. However, when constructing such an example, the protocol usually has lots of different similar attack states, and usually some of them can be recast in the \( T \) model. The difficulty is thus to find an example of a protocol with an attack state in \( R \), but no attack trace in \( T \). We could only find quite contrived examples of protocols with this property, one being displayed in Fig. 6.3. The point of this example is not to provide an interesting protocol or attack, but to demonstrate that from authentication in \( T \), we cannot conclude authentication in \( R \). More formally, the example demonstrates that the following implication is wrong:

\[ \text{authFlaw}_T(G) \Rightarrow \text{authFlaw}_R(G) \]

Consider again the NSL protocol (without key-server), but now augmented with an additional challenge-response exchange as in Fig. 6.3. Consider further the goal that \( B \) wants to authenticate \( A \) on the concatenation of nonces \( (N_A, N_B, N_C) \) (i.e. \( G = (B, B, A, A, (N_A, N_B, N_C)) \) . This goal is violated in the \( R \) model as shown by the attack in Fig. 6.3. The attack begins with a normal exchange of the first three messages between \( a \) and \( b \), after which the intruder sends the first message again to \( b \) to obtain answers \((2', 3', 5')\) with a new nonce \( n_3 \). The final two steps \( 4 \) and \( 5 \) can now be interpreted as a continuation either of steps \( 1, 2 \), or of steps \( 1', 2', 3', 5' \). (Thus, labelling the final steps as \( 4', 5' \) in the example would also be correct.) In particular, the two agents may have a different interpretation: while \( b \) could have meant step \( 4 \) as an answer to step \( 3 \), \( a \) might think it was a continuation of step \( 3' \). This discrepancy can be detected, i.e. there is a reachable state of \( R \) that contains agent \( b \) in a local state while \( a \) is not in a corresponding local state.

\[ \square \]

Figure 9: Example protocol (extension of NSL without key-server) and a trace of the protocol to demonstrate a difference in the authentication goals for \( R \) and \( T \).
For this example protocol and goal, however, the message trace model does not have any flaws. In particular considering the trace from Fig. 6.3, the messages 4. and 5. are interpreted as continuations of both the primed and unprimed session, and thus since both interpretations are in every trace, there is no discrepancy between the agents.

While neither the protocol nor the attack are of practical interest, this example demonstrates that for some goals the over-approximation is not sound, i.e. we cannot rely that safety of the over-approximated model implies safety of the original model. However, one may wonder how serious this problem is in practice: in particular, if a soundness result for authentication goals can still be obtained for a restricted class of protocols. For instance, one could require that every message of the protocol must contain a hash-value of all previous messages of the protocol run. Such a restriction would however severely limit the class of protocols that can be considered.

It is worth to note that we have discovered this problem with authentication only after formally defining the models and goals, and failure to prove the claim that $T$ has at least as many attacks as $R$. Indeed we see this as part of understanding better the models that we are working with.

### 6.4 Alternative Formulation of Authentication

The conclusions that might be drawn from the problem with authentication is that the message trace model $T$ is not suitable for authentication goals at all: for authentication we need to know which message was sent in which “context”—i.e. the local state of the honest agent sending it—while in general we cannot unambiguously recover this context from the exchanged messages alone, as we have seen in the examples.

To avoid any such problems with goals for message traces, it seems obvious to add more explicit information to the message trace model. In fact, it has become a kind of standard in protocol verification to formulate authentication using special events; these events act as a kind of interface between protocol and goals, in particular, the formulation of the goals is independent from the particularities of the protocol (like the exchanged messages). We will give here the account of weak authentication as used in the AVISPA-tool [2] which is inspired by Lowe’s non-injective agreement [21]. In both cases, there are strictly stronger forms of authentication goals, namely ones that also count successful replay of messages as an attack. This form of authentication makes sense in the set rewriting model $R$, but in $P$ and $T$, every protocol has a violation of the authentication goals, as replay is trivially possible.

For the authentication goals, there are two kinds of special events. The first, $\text{witness}(a, b, c, d)$, represents that an agent $a$ intends to execute the protocol with agent $b$ and data $d$; the identifier $c$ is used to distinguish different purposes for which the data may be meant. The second kind of event, $\text{request}(b, a, c, d)$, represents the counter part, namely that $b$ has finished his part of the protocol and now relies that there is an agent $a$ who meant to execute the protocol with $b$ and using the message $d$ for purpose $c$. Note that the intruder is unable to create such events himself or “read” them in any way. It is part of modelling the protocol, to equip it with appropriate witness and request events. The reason why this kind of formulating goals is helpful for the message trace model is that all goal relevant aspects are recorded in $\text{witness}(\cdot)$ and $\text{request}(\cdot)$ facts of the trace, and the exchanged messages can completely be ignored.

To easily integrate the new kinds of events into $R$, $P$, and $T$, without changing the models und such that we can reuse the Theorems proved about them, we make the special events part of the

---

4Proof sketch: We use the fact that the intruder knows no private keys but his own. It follows that, if an agent creates a nonce $n$ and sends it as part of a message $m$ encrypted with the public-key of an honest agent $a$, then $n$ can only appear in a later event, if the message $m$ was received and processed by $a$. Consider the situation that an honest agent $a$ in $B$ received $\{n_2\}_{pk(b)}$ and $\{n_3\}_{pk(b)}$ for two nonces $n_2$ and $n_3$ that he sent earlier as part of messages 2. and 4., i.e. he earlier sent $\{n_1, n_2, b\}_{pk(a)}$ and $\{n_1, n_3, b\}_{pk(a)}$ for some $n_1$ and $n_3$, and also $a$ has previously received the message $\{n_1, a\}_{pk(b)}$. Then $a$ must have eventually received the said messages of step 2. and 4. from $b$, namely $\{n_1, n_2, b\}_{pk(a)}$ and $\{n_1, n_3, b\}_{pk(a)}$ and answered to it. This answer must be $\{n_2\}_{pk(b)}$ and $\{n_3\}_{pk(b)}$, respectively, due to the protocol. Thus, $\text{state}_A(a, b, n_1, n_2, n_3)$ must be a subset of the trace. So when $B$ has reached a successful end of the protocol run in $T$, the $A$ must agree with it on the agent identities and nonces.

It thus holds for the example protocol that the message trace model contains no attack trace, while there is an attack state in the $P$ model. This demonstrates that $\text{authFlaw}(G)$ does not imply $\text{authFlaw}(G)$.
The difference results from the fact that in the new formalization the events witness information to get all the information from there, which was necessarily a parameter in the original formulation.

Theorem 9.

The formulation for $P$ exchanged messages. To that end, let witness($\cdot$) and request($\cdot$) be two function symbols of $\Sigma$ that are used merely for this purpose. Further, there are no intruder deduction rules for the symbols witness($\cdot$) and request($\cdot$), i.e. the intruder can neither “decrypt” such a “message”, nor construct one himself. Fig. 10 shows how the witness and request facts can be added to the messages of the protocol itself, appearing only in the message patterns of the role that creates the event. Note that $N_A$ and $N_B$ are constants used here to identify the purpose of the exchanged nonces in the protocol.

Using the new events, we can now formulate authentication for the models $R$ and $T$ in a protocol-independent way:

$$\text{authFlaw}_{R} = \exists s \in R. \exists \sigma[\text{dom} = \{A, B, P, M\}] . \text{A} \sigma \neq i \land \text{B} \sigma \neq i \land \text{request}(A, B, P, M) \sigma \in DY(s) \land \text{witness}(B, A, P, M) \sigma \notin DY(s)$$

$$\text{authFlaw}_{T} = \exists t \in T. \exists \sigma[\text{dom} = \{A, B, P, M\}] . \text{A} \sigma \neq i \land \text{B} \sigma \neq i \land \text{request}(A, B, P, M) \sigma \in DY([t]) \land \text{witness}(B, A, P, M) \sigma \notin DY([t])$$

The formulation for $P$ is identical with the one for $R$, except that $s \in P$.

We can now finally conclude that all authentication flaws of the $R$ model can also be found in the $T$ model, and thus see the completeness of the over-approximation for authentication flaws formulated over witness($\cdot$) and request($\cdot$) events:

Theorem 9.

1. $\text{authFlaw}_{R}$ implies $\text{authFlaw}_{P}$, and
2. $\text{authFlaw}_{P}$ implies $\text{authFlaw}_{T}$.

Proof.

1. Let $s \in R$ be an attack state that satisfies $\text{authFlaw}_{R}$. By Theorem 1, there is a state $s' \in P$ with $s' \supseteq s$ and $DY(s') = DY(s)$. Since both states have the same intruder knowledge, they in particular contain the same witness($\cdot$) and request($\cdot$) facts. Thus $s'$ is an attack state, and $\text{authFlaw}_{P}$ holds.

2. Let $s \in P$ be an attack state that satisfies $\text{authFlaw}_{P}$. By Theorem 2, there is a trace $t \in T$, such that $DY(s) = DY([t])$. Again, since $[t]$ has the same intruder knowledge as $s$, it contains the same witness($\cdot$) and request($\cdot$) facts, thus $[t]$ is an attack trace and $\text{authFlaw}_{T}$ holds.

\[\text{Note that in contrast to our first formalization of authentication goals, there are no parameters in this case; this difference results from the fact that in the new formalization the events witness($\cdot$) and request($\cdot$) carry enough information to get all the information from there, which was necessarily a parameter in the original formulation.}\]
Authentication in the Reachable Facts/Events Models. In the reachable facts and events models, the formulation of authentication goals is yet more difficult. The reason is that in these models, all states (or all traces) are merged into one. Suppose there is a reachable attack state $s \in \mathbb{P}$, i.e. there are certain facts $f_1$ and $f_2$, such that $f_1 \in s$ and $f_2 \notin s$. If there is a reachable state $s' \in \mathbb{P}$ such that $f_2 \in s'$, then $f_1, f_2 \notin s$, and the attack predicate would fail on $s$. Note that Theorem 4 guarantees for this case only that there is a state $s \in \mathbb{P}$ such that $f_1, f_2 \in s$, i.e. that these facts are not exclusive on all states.

We thus need different ways to formulate authentication goals in the reachable facts and events models. When data abstraction is performed, i.e. when the protocol has no fresh data anymore, then we can exploit the way the abstract terms that replace the fresh data are generated. As an example, consider again the NSL example, the goal that $A$ authenticates $B$ on $N_B$, and the data abstraction $nb(B, A)$ as the nonce created by $B$ for $A$. Then this authentication goal can be expressed as follows:

$$\exists \sigma[\text{dom} = \{A, B, M\}], A \sigma \neq i \land B \sigma \neq i \land \text{request}(A, B, N_B, M) \sigma \in \mathcal{D}\mathcal{Y}(\mathbb{F}) \land M \sigma \neq nb(B, A) \sigma$$

This goal states that it is an authentication flaw if $A$ finishes the protocol with any value other than $nb(B, A)$ for the nonce from $B$. Intuitively, the form $nb(B, A)$ of nonce $N_B$ can be seen similarly to a witness-event: $N_B$ has this form iff it has been created by $B$ for $A$ for the purpose of $N_B$. The construction of the abstract data thus allows us to declaratively see who created the data for whom and for what purpose.

More generally, consider a protocol with data abstraction (i.e. no fresh data) and labelled with witness and request facts as desired. We require that we are given a description of the abstraction by a function $\lbrack \cdot \rbrack$ that maps from variables to terms, and two functions $\text{sndr}(\cdot)$ and $\text{rcvr}(\cdot)$ from variables to variables (representing agent names). These functions express which protocol variable get abstracted into which term, and who is the creator and the intended recipient. E.g. in the NSL example we might have $\lbrack N_A \rbrack = na(A, B)$, $\lbrack N_B \rbrack = nb(B, A)$, $\text{sndr}(N_A) = \text{rcvr}(N_B) = A$, and $\text{sndr}(N_B) = \text{rcvr}(N_A) = A$.

We say that a protocol description is in accordance with the data abstraction $\lbrack \cdot \rbrack$, iff the following holds. For every reachable state of $s \in \mathbb{P}$, for every substitution $\sigma$, and for every protocol variable $V$, if $\lbrack V \rbrack \sigma$ is a subterm of $s$, then $\text{witness}(\text{sndr}(V) \sigma, \text{rcvr}(V) \sigma, V, \lbrack V \rbrack \sigma) \in s$. For instance, NSL is in accordance with $\lbrack \cdot \rbrack$, since every occurrence of $na$- or $nb$- is accompanied by a suitable witness fact. Intuitively, the accordance condition ensures everything we need to get rid of the witness-facts in the reachable facts and events models: that from the value used for protocol variable $V$, we can recognize whether there is a witness-fact for this term, i.e. whether this term was indeed produced in the required way.

We can now formulate authentication for the reachable facts model as follows:

$$\text{authFlaw}_\mathbb{F} = \exists \sigma[\text{dom} = \{V, T\}], \text{request}(\text{rcvr}(V) \sigma, \text{sndr}(V) \sigma, V, T \sigma) \in \mathcal{D}\mathcal{Y}(\mathbb{F}) \land T \sigma \neq [V] \sigma \land \text{rcvr}(V) \sigma \neq i \land \text{sndr}(V) \sigma \neq i$$

The formulation for the reachable events model is identical except when replacing $\mathbb{E}$ for $\mathbb{F}$.

Finally, we can show that also the reachable facts and events models are complete for the checking of authentication goals in this formulation:

**Theorem 10.** For a protocol with data-abstraction in accordance with abstraction function $\lbrack \cdot \rbrack$, authFlaw$_\mathbb{F}$ implies authFlaw$_\mathbb{E}$ and authFlaw$_\mathbb{E}$.

**Proof.** Let $s \in \mathbb{P}$ be an attack state. In particular, $s$ contains a request term with honest participants and without matching witness. By Theorem 4, we know that there is a subset $s' \subseteq \mathbb{F}$ of the reachable facts, with $s \subseteq s'$, thus $s'$ contains the same request fact. Note that this request fact has honest agents as its first two arguments. Towards a contradiction, assume authFlaw$_\mathbb{F}$ does not hold. Since $s'$ already contains a request-fact between two honest people, we conclude $T \sigma = [V] \sigma$ (otherwise authFlaw$_\mathbb{F}$ where true). This means that $s$ and $s'$ contain a term $[V] \sigma$, \footnote{the definition can alternatively be based on any of the other models}
which is an abstraction. By the accordance condition, we have that $s$ must contain the fact $\text{witness}(\text{sndr}(V)\sigma, \text{rcvr}(V)\sigma, V, [V]\sigma)$. This means that $s$ is not an attack state, contradicting our assumption.

\section{Conclusions}

Observe that this work does not report on any experimental results. The reason is that this paper is only concerned with the questions of modeling protocols and comparing such models, while it is not concerned with the particular techniques that are used to automatically verify protocols based on these models. However, there exists a wide variety of empirically successful analysis techniques based on the models presented in this paper [7, 9, 10, 11, 13, 18, 25, 24, 2].

We have formally proved equivalence and over-approximation relationships between several protocol models. As a representative of standard protocol models, we have used the set rewriting model $R$. We showed that it is an over-approximation of the message trace model $T$ in a certain sense. For a more precise account we have defined a persistent variant $P$ of the set rewriting model; showing that this variant lies strictly between $R$ and $T$ demonstrates that there are in fact two orthogonal kinds of over-approximation. We then considered a farther going kind of control abstraction, basically turning from reachable states (traces) to reachable facts $F$ (events $E$). We show that these models, which are often used in the context of data abstraction, are themselves over-approximations of $P$ and $T$.

We then turned to the question how to define security goals in these models. For secrecy goals, it is straight-forward that the over-approximation is appropriate in the sense that any attack against secrecy in the $R$ can be recast in the $F$, $T$, $\mathcal{F}$, and $E$ models. Thus, when we verify in one of these over-approximated models that a protocol achieves its secrecy goal, then this also holds in the original $R$ model. It turns out that the over-approximation is not appropriate in this sense for checking authentication goals: we gave an example of a protocol and authentication goal that is violated according to the $R$ model, while the same protocol and goal appear flawless in the $T$ model. However, when using auxiliary events to formulate authentication, as it is done in a number of recent approaches, the over-approximation indeed works. More precisely, we can show that verification of authentication goals in $T$ model implies there verification in the $R$ model. Finally, we have shown how to formulate authentication for $F$ and $E$ using the data abstraction.

We see the main contribution of this paper by giving a precise account of several widely used models, and from that the ability to recognize such subtle problems as the one about authentication goals, and to give a provenly correct reformulation.

Besides a better understanding of the employed models, these results also pave the way for combining methods based on these models, in particular, connecting automated verification procedures with a formalization in the theorem prover Isabelle in the style of [24]. Such a connection is motivated by the fact that a protocol verification tool is not unlikely to eventually verify a flawed protocol because of a bug in that tool, while the chance that a wrong security proof is accepted by Isabelle is much smaller. (Left aside mistakes and abstractions in the formalization itself.) The results of this work give an idea how such a connection may be possible, namely that the proof object generated by an automated verification tool would be an invariant on the set of messages that can ever be sent by an honest agent, and what messages the intruder can derive. We plan to investigate the details of such a connection as future work.

\section*{References}


