AVISPA

www.avispa-project.org

IST-2001-39252

Automated Validation of Internet Security Protocols and Applications

Deliverable 5.1: Abstractions

Abstract

For automatic protocol verification to become feasible, it is often needed to abstract away from specification details or to introduce finite or regular descriptions for infinite sets of datas. Abstractions in our protocol verification context are mappings from the original rewrite rules model into a simpler model such that every protocol flaw (of the original model) is contained in the abstract model. The converse usually doesn't hold, since, due to the simplification, we may have false positives in the abstract model even when the original model is flawless. In this deliverable, we present a number of different abstractions that we have been considering in our protocol analysis tools. More specifically, we present abstractions of the nonces and of the intruder knowledge, as well as a general approach for designing abstractions.

Deliverable details

Deliverable version: *v1.0*
Person-months required: *14*
Date of delivery: *12.01.2004*
Due on: *31.12.2003*
Classification: *public*
Total pages: *30*

Project details

Start date: *January 1st, 2003*
Duration: *30 months*
Project Coordinator: *Alessandro Armando*
Partners: *Università di Genova, INRIA Lorraine, ETH Zürich, Siemens AG*

Project funded by the European Community under the *Information Society Technologies* Programme (1998-2002)

Contents

1 Introduction

The techniques that we have formalised and implemented so far in the context of the AVISPA project already support the verification of security protocols whenever the number of sessions between the agents is finite. However, the project also aims at providing methods for verifying protocols even when this restriction is uplifted, i.e. for unbounded numbers of sessions. In general, given a protocol specification P specifying exchange of messages by roles, we are interested in the following decision problem: is the protocol correct for all its instances? If the answer is *yes*, we say the protocol is *verified*.

Known (un)decidability results. Various authors have tackled the problem of protocol verification. Here, we are only interested in the works in which no "geometrical" restrictions, such as conditions on the occurrence on variables or shape of messages, are imposed. In this case, Even and Goldreich [20], and later Heintze and Tygar [23], have proved that the problem of deciding whether a protocol is correct or not for arbitrary instances is undecidable when messages of unbounded size (depth) are exchanged. In [19], Durgin, Lincoln, Mitchell and Scedrov have further shown that the problem is undecidable if the depth of messages is bounded and if there is an infinite number of constants. They have also proved that the problem is DEXPTIME-complete if message depth is bounded *and* there is a finite number of constants.

Abstraction. A classical way to circumvent the undecidability barriers is to employ abstraction techniques, which, quoting the seminal work [18], have been developed as "a theory of discrete approximation of the semantics of computer systems mainly applied to the static analysis and verification of software." In the case of security protocols, the semantics of a protocol instance is given as an initial state and a finite set of rewrite rules independent from this initial state. Abstractions in our protocol verification context are thus mappings from the original rewrite rules model into a simpler model such that every protocol flaw (of the original model) is contained in the abstract model. The converse usually doesn't hold since, due to the simplification, we may have false positives in the abstract model even when the original model is flawless. In this deliverable, we present a number of different abstractions that we have been considering in our protocol analysis tools. More specifically, in the following sections we present abstractions of the nonces and of the intruder knowledge, which have been developed and implemented by the AVISPA group at INRIA, as well as a general approach for designing abstractions, which has been formalised by the group at ETHZ.

2 Abstraction of Nonces (INRIA)

This section describes work on the abstraction of nonces that has been carried out by the INRIA project partner. Our aim is to provide a tool whose termination is ensured and whose affirmative answers to the problem of whether a protocol is correct for all its instances is correct (on the other hand, as we remarked, we cannot avoid negative answer for correct protocols). Following [19], the minimal assumptions on the rewriting rules system that permit to have termination are:

- there is only a finite number of constants,
- messages that can be sent an unbounded number of times have to be of bounded depth.

The second restriction can be seen as a direct consequence of a strong typing of the messages. In order to meet the restriction, we abstract the data created in different runs of the protocol, namely, the nonces, and assume that there is only a finite number of different constants used. More specifically, the user specifies a finite multi-set of possible initial role instances. Each principal in a role instance can participate in as many runs as the intruder wishes him to, but the nonces in each of these runs will be the same. Given a role, it is of course possible to specify two initial instances with the same constants. In this case, the nonces created by the runs of these two instances will be different. The user may also specify the name of the nonces.

For example, in the well-known Needham-Schroeder public key protocol NSPK [15], it is possible to specify the role instances:

$$A[A : a; B : b; Ka : ka; Kb : kb; Na : na1]$$
$$A[A : a; B : b; Ka : ka; Kb : kb; Na : na2]$$
$$B[A : a; B : b; Ka : ka; Kb : kb; Nb : nb]$$

A similar framework has been employed by Blanchet [6], with the difference that he considers messages of unbounded size. The drawback, however, is that the given procedure does not always terminate. Moreover, the approach proposed by Blanchet is dedicated to secrecy problems, and cannot handle easily authentication problems. For example, protocol analysis in such settings would often report trivial replay attacks.

We handle this difficulty by considering only authentication for principals playing a finite number of sessions. The nonces created by these principals cannot be re-used at will, and we are able to get rid of most of the false attacks. As shown in Table 1, on the Clark/Jacob library [15], for example,

Protocol	Verified	Time (s)
Amended Needham-Schroeder Protocol	NA	
Needham-Schroeder with Lowe's fix	yes	271
Bilateral Key Exchange	yes	21
Carlsen's Secret Key Initiator	no	6
ISO 2 Pass Parallel Mutual Authentication	yes	4
ISO 4 pass	yes	8
ISO 5 pass	yes	25
ISO 2 pass Unilateral Authentication with Asymmetric keys	yes	1
ISO 3 pass Mutual Authentication with Asymmetric keys	yes	2
ISO 2 pass Unilateral Authentication with CCFs	yes	1
ISO 3 pass Mutual Authentication with CCFs	yes	2
ISO 2 pass Unilateral Authentication with Symmetric Keys	yes	1
ISO 3 pass Mutual Authentication with Symmetric keys	yes	1
Using Non-reversible Functions	yes	5
Woo-Lam Π	yes	887
Yahalom	yes	10

Table 1: Protocols verified with INRIA's abstraction of nonces.

out of 16 protocols on which no attack were found, our procedure run out of time in one case, wrongly reported an attack in one case, and was able to prove the correctness for the 14 remaining protocols with respect to strong authentication.

The construction of the rules is too long to be given here, but is detailed, as well as the instances considered, in [11]. A key idea was to introduce so-called *oracle rules* in order to approximate knowledge acquisition from infinitely many parallel sessions.

3 Abstraction of the Intruder Knowledge (INRIA)

This section describes work on the abstraction of the intruder knowledge that has been carried out by the INRIA project partner. The approach that we investigate, initiated by Genet and Klay [22], is to over-estimate the intruder

knowledge by using regular tree languages. This method allows one to show that some states are unreachable, and hence that the intruder will never be able to know certain terms. Regular tree-languages can be used here to effectively model the knowledge that the intruder might have acquired from previous sessions.

3.1 Tree automata approximation

We have proposed in [30] an approximation function that can be automatically generated, unlike the approximation function devised by Genet and Klay [22]. Abstractions play here an important role in order to enforce the termination of the resulting procedures. This method provides efficient algorithms for the automatic verification of security protocols with an unbounded number of sessions.

The tool Is2TiF we have worked on is based on tree automata and approximation techniques developed in [22]. Our approximation function is automatically generated by Is2TiF and is applied by Timbuk (a tree automata library designed by Thomas Genet of IRISA, Rennes) to the initial automaton representing the initial configuration of the network in order to compute an over-approximation of the network. The approximation function helps Timbuk to normalise terms which are not recognised by the current tree automaton. It has been proved that the final automaton's computation eventually terminates.

Security properties such as secrecy or authentication can then be verified. We need to express the negation of the property we want to check by a tree automaton, too. Then, we compute the intersection between the final automaton computed and the property negation automaton. Since the procedure works by over-approximation, if no flaw was detected, then the protocol is verified. However, if a flaw is detected, one cannot conclude as the flaw may be artificially introduced by the approximation process. We should then try again with a finer approximation.

3.2 Problems

Our main purpose is to combine this abstraction approach with the theorem-proving with constraint solving techniques in order to exploit the strengths of each method for the verification of security protocols. As a first attempt, we have devised a means of translating HLPSL-specifications into a standard input for Is2TiF.

More specifically, to link the languages, protocol specifications in HLPSL have to be translated into the fragment of Isabelle [31] used by Is2TiF as an

input language. For example, in Isabelle, the *Andrew Secure RPC Protocol* is specified as follows

```
Says A B {| Agent A,Crypt (shrK AB){| Nonce NAB |} |}
Says B A {| Crypt (shrK AB){| Crypt (shrK AB){| Nonce NAB |},
                                                Nonce NBA |} |}
Says A B {| Crypt (shrK AB){| Crypt (shrK AB){| Nonce NBA |}
                                                        |} |}
Says B A {| Crypt (shrK AB){| Key (Agent B) (Agent A){|
                                    Agent B |}, Nonce NBA |} |}
Says A B {| Crypt (Key (Agent B) (Agent A){| Agent B |} )
                                            {| Nonce NAB |} |}
```

while a HLPSL-specification is:

```
Protocol Andrew;
 Identifiers
   A,B:  user;
   Na,Nb,N,M: number;
   K,K2,KN: symmetric_key;
 Knowledge
   A: B,K,KN;
   B: A,K,KN;
 Messages
   1. A -> B : A,{Na}K
   2. B -> A : {{Na}KN,Nb}K
   3. A -> B : {{Nb}KN}K
   4. B -> A : {K2,N}K
   5. A -> B : {M}K2
 Session_instances
   [ A:a; B:b; K:Kab; KN:kn ];
 Intruder divert, impersonate;
 Intruder_knowledge a,b,kn;
 Goal A authenticates B on K2;
```

Note that, to ease the exposition, we here use the syntax of the "old" rather than of the "new" HLPSL [3]. However an analogous translation and analogous results apply for the new HLPSL.

In order to automatically translating HLPSL specifications into the Isabelle syntax used in Is2TiF, we had to solve the following differences:

Keys and nonces renaming. In HLPSL, the identifiers are typed but are not explicitly associated with particular agents. For example, in the

HLPSL-specification of the *Andrew Secure RPC Protocol*, the symmetric key `K` is associated neither with `A` nor with `B`. In Isabelle, this key must be explicitly denoted `shrK AB`.

Intruder knowledge. In Is2TiF, the intruder initially knows all the agents of the network, their public keys and the nonces created by untrusted agents. There is no construction in Is2TiF for stating that the intruder initially holds other information. HLPSL offers the possibility to add private keys or shared keys to the initial intruder knowledge. For example, in the *Andrew Secure RPC Protocol*, thanks to the HLPSL syntax, we may specify that the intruder initially knows a symmetric key.

Goals and non-differentiation of shared keys and nonces. In the section *Goal* of a HLPSL-specification, we specify the property (or properties) we want to check. To apply the abstraction-based approximation technique implemented in Is2TiF, the negation of this property has to be encoded by a tree automaton, which we call a *negation automaton*. Consequently, the first problem we had to tackle is to automatically generate the tree automaton corresponding to the negation of the property. The second problem is that in Isabelle specifications the shared keys cannot be distinguished (contrary to HLPSL specifications). For instance, in the HLPSL-specification of the *Andrew Secure RPC Protocol*, initially the agents `A` and `B` have two out of three shared keys (`K` and `Kn`). Unfortunately, the syntax for the negation automaton and Isabelle specification does not allow us to differentiate `K` and `Kn` in property specifications. So, the verification of a secrecy of a specific shared key cannot be performed because the negation automaton deals with the secrecy of all keys shared between `A` and `B`. The same problem may occur for nonces.

After linking Is2TiF with HLPSL, we have extended the syntax of Is2TiF for the verification of protocols dealing with *memory* (storing encrypted message or keys for later use), like the *Woo-Lam* Π protocol. This allows us to extend the class of protocols that can be verified automatically in the framework of Is2TiF (and generally by tree automata techniques).

Table 2 illustrates and points out the improvements obtained when the keys and the nonces can be distinguished. In this table the notation: *Sec* means secrecy, *Auth* means authentication. The symbol $\times$ indicates that the property is not relevant for the considered protocol, *(0)* indicates that secrecy is checked for all nonces, *(1)* indicates that secrecy is checked for a specific nonce, and *fail* indicates that the computation of the final automaton is impossible.

Protocol	Is2TiF before		Is2TiF + HLPSL	
	Sec	**Auth**	**Sec**	**Auth**
Needham-Schroeder Public Key	Non-Empty[(0)]	Non-Empty	Non-Empty[(1)]	Non-Empty
Needham-Schroeder with Lowe's fix	Checked	Checked	Checked	Checked
Andrew Secure RPC Protocol	Checked[(0)]	Checked	Checked[(1)]	Checked
Kao Chow Repeated Authentication, 1	×	Checked	×	Checked
ISO symmetric key 1-pass unilateral authentication	Non-Empty	×	Checked	×
Denning-Sacco (symmetric)	Checked	Checked	Checked	Checked
Woo-Lam Π	×	fail	×	Checked

Table 2: Comparison of protocol verifications with Is2TiF.

- In the tree automata approach there is a difficulty, when two nonces are shared by the agents A and B, since a non-empty intersection may be caused by the first or the second nonce. In our implementation the secrecy property is precisely handled since we can specify which nonce has to remain secret.

- The *Andrew Secure RPC Protocol* is now specified more precisely than in the initial syntax of Is2TiF. The main difference is that we can initialise the knowledge of the intruder by a particular key. The keys were not distinguished in the previous syntax. On the one hand, in the current specification there are two keys shared between the participants A and B. On the other hand, two nonces are generated by B for A, and currently we can distinguish these two nonces.

- One of the most interesting result is for the protocol *ISO symmetric key 1-pass unilateral authentication*. It is obvious that the protocol keeps one of the nonces secret. However, with the previous version, the intersection was non-empty, because there was a nonce which appeared in clear on the network. Now, the possibility to distinguish the nonces makes possible the secrecy to be checked for a particular nonce.

The protocols *Kao Chow Repeated Authentication, 1* and *Denning-Sacco (symmetric)* were already checkable with the tool, and the results do not change. Finally, the protocol *Woo-Lam* Π was not checkable with the previous version, but is checkable now thanks to the improvements.

3.3 Perspectives

Our main objective remains to implement for AVISPA a verification module based on the approximation function defined and implemented in Is2TiF. The

current linking does not allow verifying authentication properties nor protocols using XOR encryption. This new version will be directly based on the intermediate format IF. We intend to develop a first version using Timbuk, and then we would like to develop a tool more efficient than Timbuk in the context of protocol verification. Since the associative commutative properties of the messages sets constructor: *union* are a source of combinatorial explosion in the computation of approximations, an important objective is to circumvent this operator in future implementations. We can imagine the following scenario for analysing a protocol in the future AVISPA architecture: we first attempt to find an attack on a few protocol sessions with efficient back-ends; if no attack is reported then a verification of the protocol can be attempted using the tree automata module.

4 Abstractions in Fixed-Point Computation (ETHZ)

This section describes work on the use of abstraction in fixed-point computation, which has been carried out by the ETHZ project partner. More specifically, we present here a new approach for protocol verification that combines, extends and generalises a number of existing ideas, e.g. [9, 10, 11, 12, 14, 22, 26, 30, 31, 32]:

Persistency/Fixed-Point. We relate two protocol models, namely Paulson's inductive approach and the infinite-state transition system induced by the IF rewrite rules. We show that the main difference between the models is a kind of persistency, which we can easily integrate into IF. We show that persistency drastically simplifies our protocol model, allowing for a fixed-point computation that is independent of the interleaving of actions. Moreover, this fixed-point approach, if the fixed-point is finite, is often well-suited for the analysis of compositions of protocols: for "good" protocols the size of the fixed-point is often roughly the sum of the size of the fixed-points of its sub-protocols.

Data abstraction. We can use nonce abstraction to easily obtain a further simplified model, which is finite-state for finitely many agents. We will argue that this kind of abstraction is particularly useful in combination with the other techniques we employ, where by "useful" we mean that this kind of abstraction rarely introduces false attacks.

Symbolic representation. We introduce variables to handle also an unbounded number of honest agents. Note that this is not really an

abstraction technique but rather a quite straightforward, and efficient, extension of our methods; we have thus decided to include it here, where it can be presented in a simple and natural way.

Backward search. We finally show that it is possible to perform the fixed-point computation backward, i.e. starting from (symbolic) attack facts, trying to reach initial facts by the backward application of rules.

Hence, our contributions are:

- Combination: We combine several different techniques and argue why this combination works smoothly. This provides us with an effective method for automatic verification of security protocols.
- Generalisation: the combination builds on, and extends and generalises, many existing ideas and techniques.
- Explanation: The generalisation also provides an alternative perspective on several existing verification approaches, and thereby provides an explanation why certain simplifications of the protocol model are "useful" than other ones. In particular, this allows us to clarify the relationships between different protocol models, namely Paulson's model and the model based on persistent IF.
- If, as a result of the analysis, a protocol turns out to be flawed for a particular goal, then one might want to check the protocol again, but this time for different, weaker goals. For example, if strong authentication is violated, then one might want to check for weak authentication. A nice aspect of the fixed-point of reachable facts is that for this new analysis the fixed-point need not be computed again. In other words, the fixed-point computation is completely independent of the goals (and once the fixed-point is computed, any goal can be checked).
- We introduce a class of abstractions that are well-suited for protocol verification and we offer a wide spectrum of abstractions (i.e. a trade-off between precision and size of the fixed-point). The abstraction method is also well-suited in relation with the other techniques we use, namely persistency and fixed-point. Also, there is a simple way to choose the "next-best" abstraction in case there is a false attack.
- The new methods we have devised have led us to a number of (partially new) decidability and undecidability results, which we are currently refining and which we will report on in a forthcoming note. For example,

Role A:

$$
\begin{array}{rll}
(step\ 0) & \mathsf{state}_{\mathsf{Alice}}(1, A, B, K, \mathsf{dummy}, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(\mathsf{i}) \\
\stackrel{NA}{\Rightarrow} & \mathsf{state}_{\mathsf{Alice}}(2, A, B, K, NA, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(\{NA, A\}_{K(B)}) \\
(step\ 2) & \mathsf{state}_{\mathsf{Alice}}(2, A, B, K, NA, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(\{NA, NB\}_{K(A)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Alice}}(3, A, B, K, NA, NB, SID) & .\,\mathsf{iknows}(\{NB\}_{K(B)})
\end{array}
$$

Role B:

$$
\begin{array}{rll}
(step\ 1) & \mathsf{state}_{\mathsf{Bob}}(1, B, A, K, \mathsf{dummy}, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(\{NA, A\}_{K(B)}) \\
\stackrel{NB}{\Rightarrow} & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, NB, SID) & .\,\mathsf{iknows}(\{NA, NB\}_{K(A)}) \\
(step\ 3) & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, NB, SID) & .\,\mathsf{iknows}(\{NB\}_{K(B)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Bob}}(3, B, A, K, NA, NB, SID) & .\,\mathsf{iknows}(\mathsf{i})
\end{array}
$$

Figure 1: NSPK rules in IF.

we can show that the security question for the simplified model we have designed is decidable when restricting the size of messages (e.g. by a typed model), but undecidable otherwise. The undecidability proof is simpler than the similar one in [19], and it applies to an even further simplified protocol model.

In order to discuss all these issues in detail, let us briefly set up the context of our work by recalling some notions and introducing new ones.

4.1 Context

4.1.1 The Intermediate Format

We base our presentation on the Intermediate Format IF, which is described in detail in deliverable 2.3 [3]. Here, we only recall a few important notions at hand of the example of the *Needham-Schroeder Public Key* protocol (NSPK), which is given in Figure 1 and which will be our running example in this section. The rules given in the figure are set-rewrite rules (where "." is an associative, commutative, and idempotent operator used to construct sets) that describe the behaviour of honest agents (roles) according to NSPK. More specifically, they describe the transition relation of a transition system where each state is a set of facts like state and iknows. This transition system can be finite or infinite, depending on whether the initial state contains

(*gen step* 1) $\mathsf{iknows}(NA) \Rightarrow \mathsf{iknows}(\{NA, A\}_{K(B)})$
(*gen step* 2) $\mathsf{iknows}(NA).\mathsf{iknows}(NB) \Rightarrow \mathsf{iknows}(\{NA, NB\}_{K(A)})$
(*gen step* 3) $\mathsf{iknows}(NB) \Rightarrow \mathsf{iknows}(\{NB\}_{K(B)})$

Figure 2: Message generation rules for the intruder specialised to typed NSPK.

infinitely many state terms of honest agents. Each state term represents a partial execution of the protocol, also called a (*protocol*) *run* or *session*). We therefore also say that there is a *bounded* (respectively, *unbounded*) *number of sessions* iff the initial state contains a finite (respectively, countably infinite) number of agent facts.

We use the following syntactic sugar to simplify the presentation: since the Dolev-Yao intruder never "forgets" anything, every iknows fact is *persistent* in the sense that, if it is true in a state, then it is also true in all successor states; therefore, if an iknows fact appears on the left-hand side (LHS) of a rule, then one may omit this fact on the right-hand side (RHS) of the rule. We also call this the *persistency of intruder knowledge*.

Note also that the session id *SID* is only a technicality introduced to allow several facts in one state to be identical — up to the session id. We will see that in the simplified models we consider below we can forget about this id. (Further explanations about the syntax of the rules can be found in [3], e.g. about the existentially quantified variables, such as NA and NB, which are used to denote fresh constants, about dummy values, about key-tables, etc.)

4.1.2 Lazy Compilation

In a typed protocol model[1], we can compile the different ways the intruder can generate new messages that have the format of one of the messages in the protocol — all other messages he can compose are irrelevant as they can never be received by an honest agent. For example, in the first rule of role B of the NSPK, the intruder needs to create a message of the form $\{NA, A\}_{K(B)}$. There are essentially two ways in which he can do this: either the intruder directly has such a message in his knowledge (which he probably can not analyse) or he composes it, i.e. he knows $k(B)$, NA, and A. The message composition rules for the intruder look then as in Figure 2 and

[1]We often refer to a model with bounded messages as the *typed model*: if the agents can "magically" detect and reject messages of incorrect type (where, for example, an agent name is confused with a key), then it is not necessary to consider messages of arbitrary depth.

Role A:

$$
\begin{array}{rll}
(step\ 0) & \mathsf{state}_{\mathsf{Alice}}(1, A, B, K, \mathsf{dummy}, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(\mathsf{i}) \\
\stackrel{NA}{\Rightarrow} & \mathsf{state}_{\mathsf{Alice}}(2, A, B, K, NA, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(\{NA, A\}_{K(B)}) \\
(step\ 2) & \mathsf{state}_{\mathsf{Alice}}(2, A, B, K, NA, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(\{NA, NB\}_{K(A)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Alice}}(3, A, B, K, NA, NB, SID) & .\,\mathsf{iknows}(\{NB\}_{K(B)}) \\
(step\ 2') & \mathsf{state}_{\mathsf{Alice}}(2, A, B, K, NA, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(K(A)). \\
 & & .\,\mathsf{iknows}(NA).\mathsf{iknows}(NB) \\
\Rightarrow & \mathsf{state}_{\mathsf{Alice}}(3, A, B, K, NA, NB, SID) & .\,\mathsf{iknows}(\{NB\}_{K(B)})
\end{array}
$$

Role B:

$$
\begin{array}{rll}
(step\ 1) & \mathsf{state}_{\mathsf{Bob}}(1, B, A, K, \mathsf{dummy}, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(\{NA, A\}_{K(B)}) \\
\stackrel{NB}{\Rightarrow} & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, NB, SID) & .\,\mathsf{iknows}(\{NA, NB\}_{K(A)}) \\
(step\ 1') & \mathsf{state}_{\mathsf{Bob}}(1, B, A, K, \mathsf{dummy}, \mathsf{dummy}, SID) & .\,\mathsf{iknows}(K(B)). \\
 & & .\,\mathsf{iknows}(NA).\mathsf{iknows}(A) \\
\stackrel{NB}{\Rightarrow} & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, NB, SID) & .\,\mathsf{iknows}(\{NA, NB\}_{K(A)}) \\
(step\ 3) & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, NB, SID) & .\,\mathsf{iknows}(\{NB\}_{K(B)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Bob}}(3, B, A, K, NA, NB, SID) & .\,\mathsf{iknows}(\mathsf{i}) \\
(step\ 3') & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, NB, SID) & .\,\mathsf{iknows}(K(B)).\mathsf{iknows}(NB) \\
\Rightarrow & \mathsf{state}_{\mathsf{Bob}}(3, B, A, K, NA, NB, SID) & .\,\mathsf{iknows}(\mathsf{i})
\end{array}
$$

Figure 3: NSPK honest agent rules with the lazy compilation of messages integrated.

can be compiled into the step rules of the honest agents as in Figure 3. For protocols with more complex messages, this approach leads to a large number of rules. The advantage is that under the restriction to a typed model, the intruder rules for composing messages can now be omitted entirely and hence the closure of the intruder knowledge under intruder rules is finite.[2]

[2]Note, however, that this is not so simple for protocols with non-atomic keys. For such a protocol, one has to perform the same compilation on the analysis rules also according to the composed keys that may appear in the protocol.

4.1.3 Number of Agents and Sessions

In each rule of NSPK in Figure 1, there is exactly one state fact on both sides of the rule, so the number of agents and instances of agents that can participate in a protocol run is given by the initial state and can never change during transitions. In other words, there is a bounded number of agents and sessions iff the initial state is a finite set of facts. Note that the problem of how to represent this with a finite initial state can be solved by an appropriate initialisation theory as in [19], but we will also see that this is not necessary.

In our case, the initial state for NSPK is

$$\begin{array}{ll} & \bigcup_{A,B\in\mathsf{Agent},A\neq i,SID\in\mathbb{N}} \mathsf{state}_{\mathsf{Alice}}(1,A,B,k,\mathsf{dummy},\mathsf{dummy},SID) \\ \cup & \bigcup_{A,B\in\mathsf{Agent},B\neq i,SID\in\mathbb{N}} \mathsf{state}_{\mathsf{Bob}}(1,B,A,k,\mathsf{dummy},\mathsf{dummy},SID) \end{array}$$

where k is a constant (the key-table) and Agent is the set of all agents (which is infinite and also contains the intruder). This defines our base model to contain for every pair of agents an unbounded number of parallel sessions (distinguished by the session id *SID*). Our aim is to verify protocols without bounding the number of sessions.

4.1.4 Restrictions

The methods we present here work for a sub-language of IF. In particular, we do not allow arbitrary transitions of honest agents. Rather, we assume that all transition rules of honest agents have the form

$$\mathsf{state}_{Role}(\alpha).\mathsf{iknows}(\beta) \Rightarrow \mathsf{state}_{Role}(\gamma).\mathsf{iknows}(\delta)$$

where $\alpha, \beta, \gamma, \delta$ are arbitrary message terms and *Role* is one of the protocol's roles.

The question is how restrictive this restriction is in practice. To get some formal handle on this, we say that "in practice" at least every rule must describe one protocol step and not an internal action of the honest agent using dummy input and output. With this, we have indeed a quite restricted IF model: there is no negation, i.e. an agent reacts if his knowledge and the incoming message have a certain form, but he cannot react when the message is not of a certain form. Another restriction is that memory cannot be shared over several sessions of an honest agent.

In other words, the restriction excludes additional facts such as contains (cf. [3]), as well as negative conditions and facts. The following theorem, however, states that we can still support a large class of protocols under our restriction:

$$\frac{}{<>\in \mathcal{T}}\ (axiom)$$

$$\frac{tr \in \mathcal{T} \quad NA \notin \text{used}\, tr}{tr : (1.\ A \to B :\ \{NA, A\}_{K(B)}\ ) \in \mathcal{T}}\ (step\ 1)$$

$$\frac{tr \in \mathcal{T} \quad (1.\ \_ \to B :\ \{NA, A\}_{K(B)}\ ) \in tr \quad NB \notin \text{used}\, tr}{tr : (2.\ B \to A :\ \{NA, NB\}_{K(A)}\ ) \in \mathcal{T}}\ (step\ 2)$$

$$\frac{tr \in \mathcal{T} \quad (1.\ A \to B :\ \{NA, A\}_{K(B)}\ ) \in tr \quad (2.\ \_ \to A :\ \{NA, NB\}_{K(A)}\ ) \in tr}{tr : (3.\ A \to B :\ \{NB\}_{K(B)}\ ) \in \mathcal{T}}\ (step\ 3)$$

Figure 4: Paulson's inductive trace model for NSPK.

Theorem 1 (Compiling Alice & Bob protocols into IF). *Protocols in the variant of the Alice & Bob notation defined in [25] can be translated into the restricted IF rewrite rules.*

The theorem follows straightforwardly from the corresponding theorem in [25], by appropriately adapting/extending the language(s) considered.

4.2 Paulson's Inductive Model

As we already recalled above, in his approach to protocol verification, Paulson [31] models a protocol as an inductively defined set of traces. For example, Figure 4 shows the rules that describe the behaviour of honest agents according to NSPK.

These rules describe the protocol in terms of a set of traces, where a trace is a sequence of messages. (Note that we have introduced step-numbers into the messages for reasons explained below.) The most obvious difference to our IF model is the absence of an explicit representation of the agent states. Intuitively, the premises of each rule can be interpreted as the current state of an agent (described by the incoming and outgoing messages that would lead to this state) and an incoming message to which he can react. The outgoing message in the conclusion of the rule is the reply of the agent. Creation of fresh nonces is modelled by using an arbitrary value that does not appear in the trace yet. Note also that in Paulson's original approach, the honest agents accept only messages of correct type and format, so type-flaw attacks are excluded.

For the intruder model, we can assume the same model as in the IF, however Paulson uses a model that is restricted to atomic keys. From the modelling point of view, there is no need for this restriction (but from the theorem proving point of view it does simplify things).

$$
\begin{array}{lll}
1. & a \to b: & \{na_1, a\}_{k(b)} \\
2. & b \to a: & \{na_1, nb_1\}_{k(a)} \\
3. & a \to b: & \{nb_1\}_{k(b)} \\
3. & a \to b: & \{nb_1\}_{k(b)} \\
2. & b \to a: & \{na_1, nb_2\}_{k(a)} \\
3. & a \to b & \{nb_2\}_{k(b)}
\end{array}
$$

Figure 5: An example trace of the NSPK in Paulson's model.

4.2.1 Undesirable Traces

As shown in the example trace given in Figure 5, the model allows some traces that are somewhat questionable, which we can intuitively understand as follows: if an agent has reached a certain state S and a transition from S to S' is possible according to the protocol, then the agent can "fork" and go to *both* the states S and S'. In other words, reached agent states are *persistent*: an agent state contained in a reachable state S is contained in all successor states of S.

The reason for this remarkable property is that whenever a rule is applicable to a trace, it *remains* applicable to all extensions of the trace; only the fresh constants generated during a "transition" are different. For this reason, an infinite number of sessions can be "spawned" from every trace (even from the empty trace, and even if we bound the number of agents that can be substituted for the roles).

The persistency leads to the fact that protocols cannot be checked for strong authentication. Indeed, in this model every protocol trivially violates strong authentication: the intruder just needs to replay the final message of the protocol again and the receiver will accept it again.[3]

Note also that Paulson's model originally did not contain step numbers, which may lead to problems for some protocols. To illustrate this, consider a variant of the NSPK protocol where Alice in the final step sends not only the nonce of Bob back to him, but also again her own name, so the first and the last step of the protocol have the same format. So, without step numbers, there is nothing that would distinguish the event of Alice sending a message of either step; only in the trace we can see that the nonce used was either used before or is fresh. However, Paulson's rules ensure only that the nonces created in the application of a rule are fresh, but not that they were fresh at the time a certain event occurred in the trace. (Actually, the trace

[3]Note that in this deliverable we check different forms of authentication and secrecy, which can be expressed in terms of state and iknows facts; cf. the deliverables [2, 3].

is first transformed into a set of events and then only membership can be checked.) So, in the end, in Paulson's model Alice cannot, in general, decide if a certain nonce was created by herself or by somebody else. Note that this problem also carries over to goals. An easy way to solve the problem without changing the whole model, and in particular the fact that the rules have the form of Horn clauses, is to label each message with the step number the sender intended (and allow the intruder to send messages with any step number).

4.2.2 Persistency in IF

This observation leads to the idea that Paulson's model can be seen as the IF with *persistent state facts*: in every transition the agent goes to a new local state while "another incarnation" of the agent remains in the previous local state. Figure 6 shows the IF rules with such persistent state facts for NSPK. The session identifiers (SIDs) can be omitted now for reasons explained below (but it is safe to leave them in). The initial state (without SIDs) is simply

$$\begin{array}{ll} & \bigcup_{A,B\in\mathsf{Agent},A\neq i} \mathsf{state}_{\mathsf{Alice}}(1,A,B,k,\mathsf{dummy},\mathsf{dummy}) \\ \cup & \bigcup_{A,B\in\mathsf{Agent},B\neq i} \mathsf{state}_{\mathsf{Bob}}(1,B,A,k,\mathsf{dummy},\mathsf{dummy}) \quad . \end{array}$$

Note that this set is finite iff Agent is finite.

In [29], we formally show an equivalence between Paulson's model and this IF variant with persistent state facts for secrecy and weak authentication for protocols that can be described by IF rules of the restricted form defined in §4.1.4. The intuition behind this proof is that the premise of a rule of Paulson's model can be seen as a state fact and an incoming message. Since we will, in the following, consider only models with persistent state facts, we introduce the same syntactic sugar as for the iknows facts before: they are implicitly persistent.

4.3 Fixed-Point Computation

Paulson's model can be seen as an over-approximation of the original IF model with an unbounded number of state facts in the initial state. This over-approximation makes things simpler: in the following we will exploit the circumstance that all fact symbols are now persistent. However, this over-approximation may lead to *false positive* attacks, i.e. attacks that only work in the over-approximation, but not in the original model.

It is infeasible to directly explore the transition system described by the Paulson-inspired persistent IF-model: on every ply of the search tree, the branching degree increases by at least one due to the fact that new local

Role A:

$$
\begin{array}{rll}
(step\ 0) & \mathsf{state}_{\mathsf{Alice}}(1, A, B, K, \mathsf{dummy}, \mathsf{dummy}) & .\ \mathsf{iknows}(\mathsf{i}) \\
\stackrel{NA}{\Rightarrow} & \underline{\mathsf{state}_{\mathsf{Alice}}(1, A, B, K, \mathsf{dummy}, \mathsf{dummy})} & . \\
 & \mathsf{state}_{\mathsf{Alice}}(2, A, B, K, NA, \mathsf{dummy}) & .\ \mathsf{iknows}(\{NA, A\}_{K(B)}) \\
(step\ 2) & \mathsf{state}_{\mathsf{Alice}}(2, A, B, K, NA, \mathsf{dummy}) & .\ \mathsf{iknows}(\{NA, NB\}_{K(A)}) \\
\Rightarrow & \underline{\mathsf{state}_{\mathsf{Alice}}(2, A, B, K, NA, \mathsf{dummy})} & . \\
 & \mathsf{state}_{\mathsf{Alice}}(3, A, B, K, NA, NB) & .\ \mathsf{iknows}(\{NB\}_{K(B)})
\end{array}
$$

Figure 6: The IF rules for NSPK role A with persistent state facts. As a syntactic sugar, we will omit the underlined *implicitly persistent state facts* in the following.

agent facts are introduced with every transition and all previously present state facts remain. One could say that the persistency implies an unbounded number of sessions, as every state fact represents a partial execution of the protocol by an honest agent. That is the reason why we don't need session identifiers here to distinguish an unbounded number of sessions, as every single fact can spawn an unbounded number of sessions. (But of course it would not be wrong to also include SIDs in every state fact.)

The persistency offers a different view of the problem whether a goal state is reachable: since all facts are persistent, we might as well compute the set of reachable facts RF for a given initial state i and a set of persistent rules R:

$$
\begin{aligned}
RF_{(i,R)} &:= \bigcup_{n \in N} S_n(i, R) \\
S_0(i, R) &:= i \\
S_{n+1}(i, R) &:= S_n(i, R) \cup \bigcup_{r \in R} induc(S_n(i, R), r) \\
induc(S, (l \stackrel{v_1,\ldots,v_n}{\Longrightarrow} r)) &:= \bigcup_{\sigma \in \{\sigma \mid l\sigma \subseteq S\} \cup \mathsf{fresh}(v_1,\ldots,v_n)} r\sigma
\end{aligned}
$$

From this definition, it follows that the security question can be seen as the fixed-point of an *inductive set of facts*. However, this is not so easy to see, since we must find an appropriate renaming of the freshly generated constants. In [29] we show that a reachable state is a goal state iff some subset of the reachable facts is a goal state. The idea behind this proof is the following: given a reachable goal state, then with the same sequence of

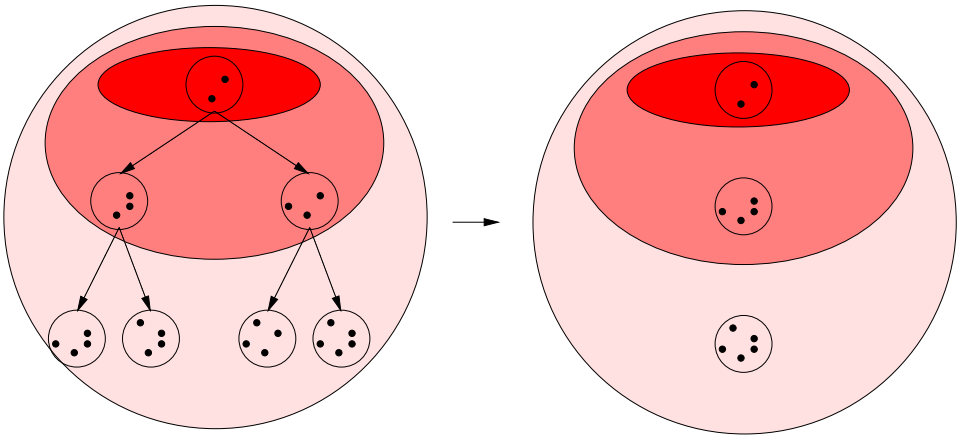

Figure 7: Fixed-point computation of facts instead of states.

transitions the respective facts can be reached in the fixed-point computation. For the converse, we first assume that every fact in the set of reachable facts is labelled with a *proof*, i.e. the name of the rule as well as the proofs of the facts from which it was first derived. We can *execute* a proof in the transition system that defines the reachable states, by performing the actions contained in the proof. Given a subset of the reachable states S that forms a goal state, we must therefore be able to obtain a reachable state S' by executing the proofs of S such that S' is equal to S modulo renaming of the fresh constants created by honest agents. Hence, S' is a goal state.

The circumstance that we can now safely check the set of reachable facts rather than the reachable states demonstrates the great simplification that the persistency of state facts has given us: as illustrated in Figure 7, we abstract away the temporal order of actions entirely and thereby get rid of the interleaving problem, i.e. the need to consider all possible permutations of the actions possible. (Note that the set of reachable facts is the same as the union of facts in reachable states under renaming of constants.)

However, at this stage, we cannot directly compute the fixed-point of reachable facts as there are still three sources of infinity in the above definition:

1. There is no bound on the size of messages that can be generated by the intruder and the honest agents. We handle this problem by using a typed model as discussed above. As we will see, this is an unavoidable restriction for our approach.

2. There is no bound on the number of fresh messages that can be created

by the honest agents. We will investigate in §4.4 an abstraction that leads to a finite representation for a bounded number of agents.

3. There is no bound on the number of agents that can participate. We will handle this problem in §4.6 using a symbolic representation that is compatible with the abstraction for the fresh messages.

4.4 Nonce Abstraction

As already remarked above, data-abstraction is a common technique in model-checking when the concrete model works with data from an infinite domain (like the natural numbers): one defines an equivalence relation on the domain with a (usually finite) *index*, i.e. partitioning the infinite domain into (finitely many) equivalence classes; then one defines the abstract model by a domain that contains for each equivalence class one representative and maps the concrete data to their abstract representatives accordingly.

Under certain assumptions, this abstract model is indeed an abstraction in the sense that if the concrete model obeys a given specification, then so does the abstract model — so if the abstract model is secure, we can conclude the same for the concrete model. One assumption from which we can infer this relation between concrete and abstract model is that on the data of the abstracted type no inequality checks are performed: if some transition is only possible for states in which two values of the abstracted type are unequal, it could happen that in the concrete model the transition is possible for some state S, while it is not possible for the abstraction of S; this means that not all abstractions of reachable states of the concrete model are reachable in the abstract model.

In the approach to protocol analysis using CSP, e.g. [10, 32], the property of a system that there are no inequality checks performed on certain types (that hold the data as opposed to other control information) is called *data independence*. We can then conclude that every equivalence relation on that data type induces an abstraction that is correct only if the concrete system is correct. Once data independence is established, we don't need to prove for every equivalence relation that we intend to use that it is indeed an abstraction.

Security protocols usually are data independent: for instance, agents check that they receive back the nonce they have created earlier, but not that a nonce that they receive is different from all nonces they have seen before.

It is immediate to see that the restricted form of IF rules we have defined in §4.1.4 guarantees data-independence for all protocols that can be described

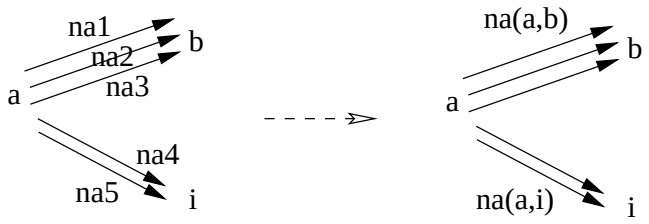

Figure 8: The intuition behind nonce abstraction.

this way: the only inequality checks are on agent names (in the goals) and we don't want to abstract on agent names.[4]

There is a large choice of possible equivalence relations on the data types. Intuitively, one could think of agents that always use the same nonce for a certain purpose (and there are only finitely many purposes) — and the protocol must be safe even then. The equivalence relation should hence represent whether the nonce was used for the same purpose. (As we will see below, however, there are good abstractions that don't meet this intuition.)

We explore here one of the most common abstractions and show a further improvement later. As a first abstraction, we use the following equivalence relation: two nonces are equivalent if they are generated by the same (honest) agent for the same communication partner(s) for the same protocol variable, e.g. agent *alice* always uses the same value for NA when talking with agent *bob*. We can thus define a set of injective functions that map every k-tuple (for a protocol with k roles) of agent names to the representative of the respective equivalence class of nonces. For instance, in NSPK, we introduce two functions na and nb meaning, for example, that $na(alice, bob)$ is the nonce that *alice* always uses for *bob* as NA. The protocol with this abstraction is shown in Figure 9.

There are several remarks in order. First, note that there are no more terms generated freshly. In particular, for a finite set of agent names, the set of nonces that can occur is also finite, and hence the state-space is finite in a typed variant — even though it represents an unbounded number of sessions. We will later show that due to this unbounded number of sessions the security question is still undecidable — despite all simplifications made so far — unless one bounds the size of messages.

Second, we have used the abstracted nonces $na(A, B)$ and $nb(B, A)$ not only in the rules where they are "created", but also in later rules in the

[4]Interestingly, one can see [16] as an abstraction on the agent names where all honest agents are abstracted into one equivalence class (which is possible since we only distinguish honest and dishonest agents).

Role A:

$$
\begin{array}{lll}
(step\ 0) & \mathsf{state}_{\mathsf{Alice}}(1, A, B, K, \mathsf{dummy}, \mathsf{dummy}) & .\,\mathsf{iknows}(\mathsf{i}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Alice}}(2, A, B, K, na(A,B), \mathsf{dummy}) & .\,\mathsf{iknows}(\{na(A,B), A\}_{K(B)}) \\
(step\ 2) & \mathsf{state}_{\mathsf{Alice}}(2, A, B, K, na(A,B), \mathsf{dummy}) & .\,\mathsf{iknows}(\{na(A,B), NB\}_{K(A)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Alice}}(3, A, B, K, na(A,B), NB) & .\,\mathsf{iknows}(\{NB\}_{K(B)})
\end{array}
$$

Role B:

$$
\begin{array}{lll}
(step\ 1) & \mathsf{state}_{\mathsf{Bob}}(1, B, A, K, \mathsf{dummy}, \mathsf{dummy}) & .\,\mathsf{iknows}(\{NA, A\}_{K(B)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, nb(B,A)) & .\,\mathsf{iknows}(\{NA, nb(B,A)\}_{K(A)}) \\
(step\ 3) & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, nb(B,A)) & .\,\mathsf{iknows}(\{nb(B,A)\}_{K(B)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Bob}}(3, B, A, K, NA, nb(B,A)) & .\,\mathsf{iknows}(\mathsf{i})
\end{array}
$$

Figure 9: Nonce abstraction for NSPK.

message and state facts. This is not a restriction of the rules, because no other term could appear in these positions. This gives rise to an idea for a further simplification/abstraction we will consider below: dropping the state facts from the rules.

Third, Figure 10 shows the attack states for which we check the fixed-point (it is possible to show that these correspond to the standard security goals, e.g. [27]). Their reading is fairly intuitive: the intruder may not find out nonces created by honest agents for honest agents (so secrecy is now a pure intruder-knowledge problem without any authentication aspects), and an honest agent may not accept any nonce to be created by another honest agent for him, unless this is indeed the case or the other agent is the intruder (who could have sent any value as his nonce). In fact, the abstraction can be seen as a labelling of the fresh nonces that says who created a nonce, for whom, and for what purpose; this makes the formulation of the goals very declarative.

Fourth, one should note the close relationship between the two abstractions that we have introduced: by persistency, all reached agent states are persistent, so the agents can continue a partial protocol run any number of times, and by the nonce abstraction they will always use the same nonces when continuing from the same point of execution.Suppose, for instance, that we take the original IF and only perform the nonce abstraction: then the same state facts (up to *SID*) can be derived any number of times. Hence,

Secrecy:

$$\begin{array}{ll} \mathsf{iknows}(na(A,B)) & B \neq i \\ \mathsf{iknows}(nb(B,A)) & A \neq i \end{array}$$

Weak authentication:

$$\begin{array}{ll} \mathsf{state}_{\mathsf{Alice}}(3,A,B,\_,na(A,B),NB) & NB \neq nb(B,A) \wedge B \neq i \\ \mathsf{state}_{\mathsf{Bob}}(3,B,A,\_,NA,nb(B,A)) & NA \neq na(A,B) \wedge A \neq i \end{array}$$

Figure 10: Goals for the NSPK under abstraction with further simplification.

the persistency abstraction can also be seen as a consequence of (this kind of) nonce abstraction.

Finally, note that for finitely many agents there can only be finitely many nonces; hence, the fixed-point of reachable states is finite in a typed model as agents will accept for every data type only messages from a finite set. This gives a simple decision procedure for the typed view of our abstracted model for finitely many agents, and, as we will show in our forthcoming note on decidability, this restriction to finitely many agents is not necessary to obtain decidability.

4.5 Related Work on Abstractions

As noted before, the abstraction of nonces by the involved participants is not new. To see the relations more clearly, let us consider one further abstraction of the model: leaving out the state facts from the rules and restricting ourselves to checking secrecy goals. (This is, of course, again an abstraction, since the applicability of rules can only increase.) The rules for NSPK would then look like Figure 11, where one can omit rule (*step* 0) and (*step* 3): the LHS of rule (*step* 0) can be added to the initial intruder knowledge, the (*step* 3) rule won't ever give any new results.

Although simplified, the rules are still meaningful in the sense that they are still an abstraction of the original model and will not introduce false attacks in many cases. For instance, (*step* 2) can be read as follows:

> If the intruder sends a message of the format of the second message of NSPK to an agent A, where the first nonce is one that A created earlier for communication with an agent B and including some other arbitrary nonce NB, then he can obtain NB encrypted with the public key of B.

One might say that all the state information that is needed in such a rule —

Role A:

$$\begin{array}{ll} (step\ 0) & \mathsf{iknows}(\mathsf{i}) \\ \Rightarrow & \mathsf{iknows}(\{na(A,B),A\}_{K(B)}) \\ (step\ 2) & \mathsf{iknows}(\{na(A,B),NB\}_{K(A)}) \\ \Rightarrow & \mathsf{iknows}(\{NB\}_{K(B)}) \end{array}$$

Role B:

$$\begin{array}{ll} (step\ 1) & \mathsf{iknows}(\{NA,A\}_{K(B)}) \\ \Rightarrow & \mathsf{iknows}(\{NA,nb(B,A)\}_{K(A)}) \\ (step\ 3) & \mathsf{iknows}(\{nb(B,A)\}_{K(B)}) \\ \Rightarrow & \mathsf{iknows}(\mathsf{i}) \end{array}$$

Figure 11: Nonce abstraction for NSPK, leaving out the state facts.

that A once created $na(A,B)$ for communication with B — is contained in the abstract nonce-term itself. Lowe's attack on NSPK consists of basically applying this rule for $B = i$ and $NB = nb(B',A)$ for some honest agent B'.

Note also that for a bounded number of agents, instantiating all the rules appropriately, we obtain the basis of the *oracle rules* of §2 (and of [11, 12, 14]). In this view, the analysis problem (for secrecy goals) has been transformed into a pure problem about the intruder knowledge. This can be interesting for several reasons as one may combine this with other kinds of analysis, namely the lazy intruder technique [1, 5, 7, 8, 13, 17, 21, 24, 28]; one may see the rules obtained from the abstraction process just as an addition to the normal Dolev-Yao intruder. This enables the use of such rules in "normal" protocol model-checking with a bounded number of sessions. The classical example is a server that is abstracted into an oracle rule for the intruder while the rest of the protocol is handled in the usual way.

Also, in [22], Genet and Klay use a finer nonce abstraction and represent the intruder knowledge by a tree automaton; the nonces are also more refined in their case. We will discuss refinements of nonce abstractions below.

To summarise what we have so far, we can say that we started from a different perspective and ended up with a generalisation of existing state-of-the-art approaches in automated protocol verification in two regards: first, we're not limited to modelling intruder knowledge but rather can also consider the agent states and thereby have a clear basis for the analysis of authentication properties, not only secrecy; second, considering the abstracted nonces as a function of several agents (which was also done in some other approaches) gives us a clearer view of the problem than when using instantiated constants

Role B:

$$
\begin{array}{lll}
(step\ 1) & \mathsf{state}_{\mathsf{Bob}}(1, B, A, K, \mathsf{dummy}, \mathsf{dummy}) & .\,\mathsf{iknows}(\{NA, A\}_{K(B)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, nb(B, A, NA)) & .\,\mathsf{iknows}(\{NA, nb(B, A, NA)\}_{K(A)}) \\
(step\ 3) & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, nb(B, A, NA)) & .\,\mathsf{iknows}(\{nb(B, A, NA)\}_{K(B)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Bob}}(3, B, A, K, NA, nb(B, A, NA)) & .\,\mathsf{iknows}(\mathsf{i})
\end{array}
$$

Figure 12: Skolem nonce abstraction for NSPK. Only role Bob is shown since for Alice the abstraction is the same.

for a finite number of agents.

Besides being more general, this gives an alternative view of the problems and both a clarification and justification of the particular simplifications and abstractions that are implicit in many approaches. This allows us, more specifically, to see our protocol model not as an extension of the Dolev-Yao intruder, but rather as an extension of the honest agent behaviour.

4.6 Symbolic Handling of the Agents

The only bound we have not addressed so far, is the bound on the number of honest agents that can participate in protocol sessions. As we discuss in more detail in deliverable 3.3 [4], we can overcome this bound by employing a symbolic representation for agent names.

4.7 Refined Abstractions

When we find a flaw in the abstracted model that does not occur in the original model (or, at least, we cannot find it in the original model), then we need to use a different abstraction that does not map so many different things into one.

We have used an injective function that maps a pair of agent names to constants. Hence, an honest agent in this abstraction uses the same nonces whenever he talks to the same communication partners. To make the abstraction "less stereotypical" (i.e. to differentiate more nonces), we hence change this functionality a bit: for the creation of fresh values of nonce NB we choose an injective function that maps a pair of honest agents *and a nonce* to a constant:

$$NB = nb(B, A, NA)$$

We call this *Skolem nonce abstraction*. Figure 12 shows the NSPK rules for role B under Skolem nonce abstraction; the rules for Role A do not change

with respect to Figure 9 as the nonce NA does not "depend" on NB. We then have:

Theorem 2. *The model with Skolem abstraction is attack-equivalent to Paulson's model for secrecy and authentication.*

Proof. (Sketch) The abstraction is a function of all the values it depends on (i.e. that were existentially introduced before). This introduces a translation (from reached facts to messages) along which the attack can then be simulated in Paulson's model. □

The refined abstraction obviously fixes the problem with the authentication, but it raises a new problem: the set of constants (and therefore also the set of reachable symbolic facts) that can occur under this abstraction is infinite. For instance, for NSPK, the facts

$$\begin{array}{l}\mathsf{iknows}(nb(B,i,na(A,i))),\\ \mathsf{iknows}(nb(B,i,nb(B,i,na(A,i)))),\\ \mathsf{iknows}(nb(B,i,nb(B,i,nb(B,i,na(A,i))))),\\ \ldots\end{array}$$

are reachable.

Obviously, such facts cannot make any difference. Indeed, whenever a fact $\mathsf{iknows}(nb(B,i,nb(X)))$ matches the LHS of an honest agent rule for any message X, then also $\mathsf{iknows}(nb(B,i,nb(B,i,na(A,i))))$ applies. (The honest agents don't check the messages in more depth, so to speak.)

Taking this idea further, we can identify a large number of messages that don't make any difference in our simplified model. In fact, we don't need to distinguish all nonces based on different incoming nonces (e.g. the nonce NA that the agent b received and upon which he generated NB) but only whether the incoming message was *genuine* or not, i.e. if it was the right (abstract) value according to the protocol. For the NSPK we have

$$\begin{array}{ll} na(A,B) & \\ nb(B,A,g_{NA}) & \text{with } g_{NA} \equiv (NA = na(A,B))\ , \end{array}$$

where g_{NA} is Boolean.

We call this abstraction *genuine Skolem nonce abstraction*. This abstraction may appear counter-intuitive: the nonces "contain" the information if they were created on the base of right data. It seems as if we are using an implicit assumption in our abstraction, namely authentication properties that we are going to show only later. But in fact this is not what we are doing here. Rather, we are merely partitioning the set of nonces that are created

Role B:

$$
\begin{array}{ll}
(step\ 1) & \mathsf{state}_{\mathsf{Bob}}(1, B, A, K, \mathsf{dummy}, \mathsf{dummy}).\mathsf{iknows}(\{NA, A\}_{K(B)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, nb(B, A, NA = na(A, B))). \\
& \mathsf{iknows}(\{NA, nb(B, A, NA = na(A, B))\}_{K(A)}) \\
 & \\
(step\ 3) & \mathsf{state}_{\mathsf{Bob}}(2, B, A, K, NA, nb(B, A, NA = na(A, B))). \\
& \mathsf{iknows}(\{nb(B, A, NA = na(A, B))\}_{K(B)}) \\
\Rightarrow & \mathsf{state}_{\mathsf{Bob}}(3, B, A, K, NA, nb(B, A, NA = na(A, B))).\mathsf{iknows}(\mathsf{i})
\end{array}
$$

Figure 13: Genuine Skolem nonce abstraction for NSPK.

by B for A as NB of our first abstraction, i.e. $nb(B, A)$, into two classes: $nb(B, A, true)$ for those that are sent in response to the incoming nonce $na(A, B)$, and $nb(B, A, false)$ for any other incoming nonce. The model is simply more fine-grained than the first one according to the concrete events happening.

Figure 13 shows the role B of the NSPK for genuine Skolem nonce abstraction. Note that we have extended the IF syntax with the equality check $NA = na(A, B)$; the semantics of this extension is simply that when applying the rule this expression is evaluated and replaced with the resulting truth-value.

Note also that the fixed-point is always finite under the genuine Skolem nonce abstraction. In the examples of the protocols NSPK/NSL and SHARE, we can now easily check secrecy and authentication; we are currently testing other examples.

4.8 Perspectives

As we remarked above, the new methods we have just presented have led us to a number of (partially new) decidability and undecidability results, which we are currently refining and which we will report on in a forthcoming note. Moreover, we are currently implementing the different abstractions described above in order to scale the applicability of our verification tools to the industrial-strength protocols.

References

[1] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In C. Palamidessi, editor, *Proceedings of Concur'00*, LNCS 1877, pages 380–394. Springer-Verlag, 2002.

[2] AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language. Available at `http://www.avispa-project.org`, 2003.

[3] AVISPA. Deliverable 2.3: The Intermediate Format. Available at `http://www.avispa-project.org`, 2003.

[4] AVISPA. Deliverable 3.3: Session Instances. Available at `http://www.avispa-project.org`, 2003.

[5] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In E. Snekkenes and D. Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, 2003. Available at `http://www.avispa-project.org`.

[6] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of CSFW'01*, pages 82–96. IEEE Computer Society Press, 2001.

[7] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP'01*, LNCS 2076, pages 667–681. Springer-Verlag, 2001.

[8] M. Boreale and M. G. Buscemi. A framework for the analysis of security protocols. In *Proceedings of CONCUR 2002*, LNCS 2421, pages 483–498. Springer-Verlag, 2002.

[9] L. Bozga, Y. Lakhnech, and M. Perin. Pattern-based abstraction for verifying secrecy in protocols. In *Proceedings of TACAS 2003*, LNCS 2619. Springer-Verlag, 2003.

[10] P. Broadfoot, G. Lowe, and A. Roscoe. Automating data independence. In *Proceedings of Esorics 2000*, LNCS 1895, pages 175–190. Springer-Verlag, 2000.

[11] Y. Chevalier. *Résolution de problèmes d'accessibilité pour la compilation et la validation de protocoles cryptographiques.* Phd, Université Henri Poincaré, Nancy, December 2003.

[12] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Extending the Dolev-Yao Intruder for Analyzing an Unbounded Number of Sessions. In M. Baaz, editor, *Proceedings of CSL'2003*, LNCS 2803. Springer-Verlag, 2003. Available at `http://www.avispa-project.org`.

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

[14] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In E. Brinksma and K. G. Larsen, editors, *Proceedings of CAV'02*, LNCS 2404, pages 324–337. Springer-Verlag, 2002.

[15] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL: `www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.

[16] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proceedings of ESOP'2003*, LNCS 2618, pages 99–113. Springer-Verlag, 2003.

[17] R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proceedings of SAS 2002*, LNCS 2477, pages 326–341. Springer-Verlag, 2002.

[18] P. Cousot. Abstract interpretation. *Symposium on Models of Programming Languages and Computation, ACM Computing Surveys*, 28(2):324–328, June 1996.

[19] N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[20] S. Even and O. Goldreich. On the security of multi-party ping pong protocols. In *Proceedings of 24th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 1983.

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

[22] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proceedings of CADE'00*, LNCS 1831, pages 271–290. Springer-Verlag, 2000.

[23] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, 1996.

[24] A. Huima. Efficient infinite-state analysis of security protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[25] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of LPAR 2000*, LNCS 1955, pages 131–160. Springer-Verlag, 2000.

[26] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre. Incremental verification by abstraction. In *Proceedings of TACAS 2001*, LNCS 2031. Springer-Verlag, 2001.

[27] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, pages 31–43. IEEE Computer Society Press, 1997.

[28] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the ACM Conference on Computer and Communications Security CCS'01*, pages 166–175, 2001.

[29] S. Mödersheim. Equivalences between Protocol Models. Manuscript, 2003.

[30] F. Oehl, G. Cécé, O. Kouchnarenko, and D. Sinclair. Automatic approximation for the verification of cryptographic protocols. In *Proceedings of Conference on Formal Aspects of Security*, LNCS 2629. Springer-Verlag, 2003.

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

[32] A. Roscoe and P. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7:147–190, 1999.