The Open-source Fixed-point Model Checker for Symbolic Analysis of Security Protocols

Sebastian Mödersheim[1] Luca Viganò[2]

[1] IBM Zurich Research Laboratory, Switzerland, smo@zurich.ibm.com
[2] Department of Computer Science, University of Verona, Italy, luca.vigano@univr.it

Abstract. We introduce the Open-source Fixed-point Model Checker OFMC for symbolic security protocol analysis, which extends the On-the-fly Model Checker (the previous OFMC). The native input language of OFMC is the AVISPA Intermediate Format IF. OFMC also supports AnB, a new Alice-and-Bob-style language that extends previous similar languages with support for algebraic properties of cryptographic operators and with a simple notation for different kinds of channels that can be used both as assumptions and as protocol goals. AnB specifications are automatically translated to IF.
OFMC performs both protocol falsification and bounded session verification by exploring, in a demand-driven way, the transition system resulting from an IF specification. OFMC's effectiveness is due to the integration of a number of symbolic, constraint-based techniques, which are correct and terminating. The two major techniques are the lazy intruder, which is a symbolic representation of the intruder, and constraint differentiation, which is a general search-reduction technique that integrates the lazy intruder with ideas from partial-order reduction. Moreover, OFMC allows one to analyze security protocols with respect to an algebraic theory of the employed cryptographic operators, which can be specified as part of the input. We also sketch the ongoing integration of fixed-point-based techniques for protocol verification for an unbounded number of sessions.

1 Introduction

The automated analysis of security protocols is a field in the intersection of formal methods and IT security that has been intensively studied during the last 20 years, e.g. [1,2,4,5,6,13,15,18,20,21,24,25,27,29,37,43,49,51,53,59,61,62,63]. The Open Source Fixed-point Model Checker OFMC, the successor of the On-the-Fly Model-Checker [12,54,58], is a freely available[3] tool that integrates the most successful techniques of this field. In this paper, we summarize its main modeling and verification techniques, pointing to the corresponding publications where the formal details and proofs can be found.

[3] OFMC is available at www.avantssar.eu together with the other back-ends of the AVISPA Tool and of the AVANTSSAR Platform.

The "native" input language of OFMC is the *AVISPA Intermediate Format IF* [7,12,54], which describes a security protocol as an infinite-state transition system using set-rewriting. OFMC also supports a simple and intuitive Alice-and-Bob-style language: *AnB* [56,57]. AnB specifications are automatically translated to IF—this translation defines a formal semantics for AnB in terms of IF. With respect to previous similar languages, AnB supports the specification of protocols that can only be executed correctly when taking the algebraic properties of cryptographic operators into account. For instance, protocols based on the Diffie-Hellman key exchange only make sense in a model where $g^{xy} \approx g^{yx}$.

Moreover, AnB allows one to specify properties of channels used for the transmission of protocol messages, namely authentic, confidential and secure channels [57]. We can specify channels both

- *as assumptions*, i.e. when a protocol relies on channels with particular properties for the transmission of some of its messages, and
- *as goals*, i.e. when a protocol is supposed to establish a certain kind of channel.

This gives rise to an interesting question: given that we have verified that a protocol P_2 provides its goals under the assumption of a particular kind of channel, can we then replace the assumed channel with an arbitrary protocol P_1 that provides such a channel? In general, the answer is negative, while we have proved in [57] that under certain restrictions such a compositionality result is possible. We also have generalized all our results to channels where agents may be identified by pseudonyms rather than by their real names.

OFMC performs both protocol falsification (i.e. detecting attacks) and bounded session verification by exploring, in a demand-driven way, the transition system resulting from an IF specification. OFMC's effectiveness is due to the integration of a number of symbolic, constraint-based techniques, which are correct and terminating. The two major techniques are the

- *lazy intruder* [12], which is a symbolic representation of the intruder, and
- *constraint differentiation* [58], which is a general search-reduction technique that integrates the lazy intruder with ideas from partial-order reduction.

Both techniques significantly reduce the search space associated to a given protocol specification without excluding attacks (or introducing new ones).

Moreover, OFMC allows one to analyze security protocols with respect to an algebraic theory of the employed cryptographic operators, which can be specified as part of the input [11]. We also sketch the ongoing integration of fixed-point-based techniques for protocol verification for an unbounded number of sessions [55].

We proceed as follows. In Section 2, we summarize the input languages AnB and IF, introducing a running example, and describe our standard protocol model in the presence of an active intruder. In Section 3, we introduce the constraint-based analysis techniques, and we then summarize our ongoing work on integrating over-approximation techniques in Section 4. In Section 5,

`Protocol` : *Authenticated Diffie-Hellman key exchange*
`Types` :
  `Agent` A, B;
  `Number` g, X, Y, Msg;
`Knowledge` :
  A : A, B, g;
  B : B, g;
`Actions` :
  $A \bullet\!\!\rightarrow B : \exp(g, X)$
  $B \bullet\!\!\rightarrow A : \exp(g, Y)$
  $A \rightarrow B : \{\!|A, Msg|\!\}_{\exp(\exp(g,X),Y)}$
`Goals` :
  $A \bullet\!\!\rightarrow\!\!\bullet B : Msg$

Fig. 1. AnB specification of an authenticated Diffie-Hellman key exchange.

we report on experimental results and focus on a major example protocol that we have analyzed with OFMC. We conclude, in Section 6, with an outlook on future work.

2 Input Languages and Modeling

In this section, we discuss the specification languages on which OFMC is based and how one can employ them to model security protocols and their properties. We first present AnB (Section 2.1) and then the AVISPA Intermediate Format IF (Section 2.2), to which AnB is internally translated (Section 2.3).

2.1 AnB

A simple and intuitive way to describe security protocols is to use the *Alice and Bob* notation, which describes how messages are exchanged between honest agents acting in the different protocol roles. This popular notation is usually used informally, but there are several formal protocol specification languages based on the Alice and Bob notation, e.g. [7,32,46,49,52,56]. We give an overview of the most advanced one here, called *AnB* [56,57].

The novel features of AnB are its support for protocols that require algebraic properties for the protocol execution, as well as a notion of several types of communication channels that can be used both as assumptions and as goals of a protocol. AnB is one of the specification languages that OFMC accepts as input. The semantics of AnB is formally defined by translation to the more low-level specification language *AVISPA Intermediate Format IF* [7]. IF is more expressive than AnB, but harder to use. It is, however, well suited for formal analysis tools such as OFMC and the other back-ends of the AVISPA Tool [5].

Fig. 1 shows a simple example protocol in AnB; we will use this protocol, which is based on the Diffie-Hellman key exchange, as a running example throughout the paper. An AnB specification comprises of 5 sections. We first state the name of the protocol and then declare the type of each identifier of the protocol specification; this is needed in the translation to IF, although OFMC can optionally ignore the types during analysis so to detect type-flaw attacks. We distinguish two kinds of identifiers, using a naming convention similar to Prolog: identifiers starting with an upper-case letter are called *protocol variables* and are instantiated during the protocol execution, whereas identifiers starting with a lower-case letter represent global constants and functions. Protocol variables of type `Agent` are called *roles*. In our example, we have the roles A and B, which get instantiated by arbitrary agents when executing the protocol. The numbers g, X, and Y are the group and the random exponents used in the Diffie-Hellman key exchange.

The next section of an AnB specification describes the initial knowledge attached to each role. We require that all variables that occur in the initial knowledge section are of type `Agent`. They will later be instantiated arbitrarily with agent names. The initial knowledge is essential for the semantics, as the way honest agents can construct the messages of the protocol depends on it. Variables that do not occur in the initial knowledge of any role represent values that are freshly created by the agent who first uses them.[4] In the example, X and Msg are created by A and Y is created by B.

The core of the specification is the list of exchanged messages, which describes the ideal protocol run without interference from the intruder. Every action in the list is of the form $A \to B : M$, meaning that the agent in the role A sends the message M to the agent in the role B. Additionally, we employ the "bullet" ("$\bullet$") notation from [50] to denote that we have a channel that ensures the identity of the respective end-point. This gives rise to the following four kinds of channels.

- *Insecure channel*: $A \to B : M$ represents an insecure channel from A to B. Insecure channels are controlled by the intruder, i.e. he can read all messages and insert messages under any sender name.
- *Authentic channel*: $A \bullet\!\!\to B : M$ represents an authentic channel from A to B. This means that B can rely on that fact that A has sent the message M and that A's intention was to send it to B. There is, however, no guarantee of confidentiality, i.e. anybody may see M.
- *Confidential channel*: $A \to\!\!\bullet\, B : M$. This means that A can rely on that fact that only B can see the message M. There is, however, no guarantee of authenticity, i.e. anybody could have sent M.
- *Secure Channel*: $A \bullet\!\!\to\!\!\bullet\, B : M$. This is a channel that is both authentic and confidential.

The behavior of channels is described more formally in Section 2.4.

[4] As a consequence, all long-term keys in the initial knowledge need to be represented as functions of agent names; for instance, one may use $sk(A, B)$ as the shared key of A and B.

In the running example, A and B generate random values X and Y and exchange the Diffie-Hellman *half keys* $\exp(g, X)$ and $\exp(g, Y)$ over authentic channels; we omit the modulus in our notation for simplicity. The final message is a pair (denoted simply by ",") consisting of A's name and a payload message *Msg* (modeled as a random number). The pair is encrypted symmetrically using the new, agreed-upon Diffie-Hellman key $\exp(\exp(g, X), Y)$, where we use $\{\!|\cdot|\!\}$. to denote symmetric encryption.

Readers unfamiliar with the Diffie-Hellman key exchange may wonder how A can actually construct this key. In fact, this is one of the main problems of previous Alice-and-Bob-style languages: the interpretation of protocols that are based on algebraic properties of the employed cryptographic operators. In AnB, this problem is solved generically with respect to arbitrary algebraic theories.[5] In this example, A knows X, which she generated herself, and $\exp(g, Y)$, which she received from B. She can thus generate $\exp(\exp(g, Y), X)$, which is equal to $\exp(\exp(g, X), Y)$ under the laws of exponentiation. The equations characterizing exponentiation are thus critical, since if we do not take them into account, then the protocol cannot even be "executed": it is unclear what an honest A should do to form the final message of the protocol. We return to this in more detail below. We assume throughout the paper a given algebraic theory defined by a set of equations and we interpret terms in the *quotient algebra* induced by these equations (see [10], for instance): intuitively, two terms are equal, denoted by $s \approx t$, iff this is a consequence of the algebraic equations.

In the final section of an AnB specification, we specify the goals that the protocol is supposed to achieve, in this case the goal that the payload message is transmitted over a secure channel from A to B. We may thus rephrase this protocol and its goal as follows: the Diffie-Hellman key exchange allows us to obtain a secure channel out of authentic channels. We have a similar setup in TLS/SSL, for instance, but we have selected the Diffie-Hellman example for brevity.

2.2 IF

We now give an overview of the AVISPA Intermediate Format IF [7], the "native" language of OFMC, and then sketch how AnB is translated into IF. IF can be considered as a kind of assembly language, because it is a low-level technical language. While it is not simple for human users to specify complex protocols in such a language, it is well-suited for automated tools as it describes the behavior of honest and dishonest agents unambiguously.

An *IF specification* $P = (I, R, G)$ consists of an *initial state* I, a *set* R *of rules* that induces a transition relation on states, and a *set* G *of "attack rules"* (i.e. *goals*) that specify which states count as attack states. The verification question is then whether an attack state is reachable from an initial state: as expected, a protocol is called *safe* when no attack state is reachable from the initial state using the transition relation. The transition system induced by I

[5] The implementation currently supports exponentiation and exclusive or.

and R usually has an infinite number of states and the verification question is in general undecidable [35,36].

Terms and States In IF, we distinguish two kinds of terms. First, we have *message terms* as we have them in AnB, with the same convention that constants start with a lower-case letter and variables with an upper-case letter.[6] On top of that, we have *facts*, which are built using distinguished function symbols and have message terms as arguments. For instance, $\mathsf{iknows}(m)$ denotes the fact that the intruder knows the message m, and $\mathsf{state}_{\mathcal{R}}(m_1, \ldots, m_k)$ is the fact that an agent has reached a local state of its execution that is characterized by the list of messages $m_1, \ldots, m_k$; this list usually consists of the agent's initial knowledge as well as previously sent and received messages. The additional parameter $\mathcal{R}$ represents the protocol role that the agent is playing (we denote all roles with calligraphic letters). Despite the upper-case convention, the role names like $\mathcal{R}$ are constants. Such constants do not appear in the AnB specification, where we have variables like A and B to denote the roles. In IF states, however, these variables will be instantiated with concrete agent names like a and b. In order to specify the role names, we thus need to distinguish in IF between variables that will hold the concrete agent names and constant identifiers for the roles that will not be instantiated. We will later introduce further fact symbols.

An *IF state* is a set of facts, separated by dots (“.”). We call a term *ground* when it does not contain variables, and an IF state is ground when all of its terms are. The transition system defined by an IF specification consists of only ground states: the initial state is ground and transitions (as we will define them below) cannot introduce variables. Later on, however, we will also consider symbolic techniques that deal with non-ground states.

Initialization In general, the security of a protocol should be defined based on an arbitrary number of agents who can execute an arbitrary number of sessions in any role of the protocol in parallel. Often, however, we consider only a fixed, bounded number of sessions, which implies also a bound on the number of agents who can participate. We can specify this directly, by concrete initial local states of honest agents, e.g. writing

$$\mathsf{state}_{\mathcal{A}}(a, 0, id17)$$

to represent an honest agent a playing role $\mathcal{A}$ at the beginning of the protocol execution (step 0), and with a unique identifier (to allow for several parallel sessions). We will later also allow that instead of the constant a, we may have a variable A (of type `Agent`) that can be instantiated by any agent name. This enables us to specify abstractly a number of sessions without enumerating concrete instances.

[6] In the concrete syntax of IF, some notation like $\{|m|\}_k$ is replaced by prefix symbols such as $\mathsf{scrypt}(k, m)$, but, for readability, we will use the pretty notation in this paper anyway.

The initial state also contains the initial knowledge of the intruder in form of $\mathsf{iknows}(m)$ facts for every "public" information m such as public keys, public constants, and also usually the names of all agents. Moreover, the intruder should be able to participate in sessions like any other agent, and therefore may have appropriate keys, e.g. public and private keys as well as symmetric keys shared with other agents.

Transition Rules We consider here IF transition rules of the following form:[7]

$$L \mid EQ \;\text{=}[V]\Rightarrow\; R$$

where L and R are sets of facts, EQ is a set of equations on terms, and V is a list of variables that do not occur in L or EQ; moreover, R may only contain variables that also occur in L, EQ or V. The semantics of this rule is defined by the state transitions it allows: we can get from a state S to a state S' with this rule iff there is a substitution σ of all rule variables such that

- $L\sigma \subseteq S$,
- $S' = (S \setminus L\sigma) \cup R\sigma$,
- $V\sigma$ are fresh constants (that do not appear in S), and
- all equations of EQ are satisfied under σ.

All equalities between terms/facts are modulo the considered algebra. The conditions on the variables ensure that S' is ground whenever S is.

As an example, consider the rule:

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

This allows the intruder to deduce the plaintext M of a symmetrically encrypted message $\{|M|\}_K$ whenever he knows the corresponding key K. Observe that the left-hand side facts are repeated on the right-hand side as otherwise the matched instances would get removed during the transition and thus the intruder would "forget" the encrypted message and the key, which, of course, we do not want. As it is safe to assume that the intruder never forgets any message, we allow the simplification that the $\mathsf{iknows}(\cdot)$ facts are *persistent*, i.e. no $\mathsf{iknows}(\cdot)$ fact gets removed during transitions so we do not explicitly need to repeat it on the right-hand side. Thus, we can simplify the previous rule to

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

Moreover, we can describe the corresponding symmetric encryption rule of the intruder simply as:

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

[7] We have here simplified the form of the rules for the ease of presentation; in [7,54] rules may contain further conditions about the inequality of terms and negative facts, which we do not need for the examples in this paper.

This rule alone produces an infinite state transition system, because the intruder can arbitrarily encrypt messages he knows ad infinitum. It is thus a particular challenge to analyze protocols in the presence of such an infinitary intruder model without excluding attacks. Note that the IF does not prescribe an intruder model—the user may specify any set of such rules—but the analysis techniques we will introduce later constrain the class of intruder models that can be considered.

The IF allows us to specify a wide variety of intruder models by giving a set of intruder deduction rules (in the style of Dolev and Yao [34]) that formalize how he can compose and decompose messages, as in the above example transitions. We slightly generalize standard intruder models in which the intruder acts only under one identity i: in our model, the intruder may have several names that he controls, in the sense that he has the necessary long-term keys to actually work under a particular name. This reflects a large number of situations, like an honest agent who has been compromised and whose long-term keys have been learned by the intruder, or when there are several dishonest agents who all collaborate. This worst case of a collaboration of all dishonest agents is simply modeled by one intruder who acts under different identities. We thus simply assume that $\mathsf{dishonest}(A)$ holds for any dishonest agent name A in the initial state (i.e. only for $A = i$ in the classical intruder model). In general, we can allow IF rules that model the compromise of an agent A or the creation of a new dishonest identity A, where we have the predicate $\mathsf{dishonest}(A)$ on the right-hand side. We also have a predicate $\mathsf{honest}(A)$ and we ensure that for every agent A either $\mathsf{honest}(A)$ or $\mathsf{dishonest}(A)$ holds.

Goals We describe the goals of a protocol by *attack states*, i.e. states that violate the goals, which are in turn described by *attack rules*. That is, in IF we describe attack states by means of rules without a right-hand side: a state at which the attack rule can fire is thus an attack state.

We give an example for the common *secrecy* goal. To that end, assume that the transition rules contain the fact $\mathsf{secret}(M, B)$ whenever an honest agent A generates a message M that is supposed to be secret with another, not necessarily honest, agent B. Thus, it is an attack if the intruder finds out M but B is honest:

$$\mathsf{secret}(M, B).\mathsf{iknows}(M).\mathsf{honest}(B)$$

We can define other standard goals like *authentication* in a similar way; see [57] for an overview.

2.3 From AnB to IF

The core of the translation from AnB to IF, i.e. the AnB semantics, is to define a "program" for each role of the protocol. This is described in IF by rules of the

form

$$\begin{aligned}&\mathsf{state}_{\mathcal{R}}(m_0,\ldots,m_k).\mathsf{iknows}(m_{k+1})\\ &=\![V]\!\Rightarrow\\ &\mathsf{state}_{\mathcal{R}}(m_0,\ldots,m_{k+2},V).\mathsf{iknows}(m_{k+2})\end{aligned}$$

where m_0 is the initial knowledge of role $\mathcal{R}$, $m_1,\ldots,m_k$ is the sequence of messages that $\mathcal{R}$ has sent and received so far, m_{k+1} is the message that $\mathcal{R}$ receives in this transition, m_{k+2} is the message that $\mathcal{R}$ replies with, and V is the set of fresh variables in m_{k+2}. This rule is applicable whenever an agent playing the role $\mathcal{R}$ is in an appropriate state and receives an appropriate message from the intruder. This reflects an optimization for the case of insecure channels: we can identify intruder and network for insecure channels (that are controlled by the intruder, see [54] for a soundness proof). If[8] we apply the rule, then the agent creates the new variables V and sends the outgoing message m_{k+2} to the "network", and also updates its local state by the received message and the sent one, and by the fresh variables. In the case of the first or the last message of the protocol, the incoming or outgoing message is omitted in the rule.

Example 1. According to this schema, the IF transition rules of A for the AnB example in Fig. 1 are as follows—ignoring the authentic channels for now:

$$\begin{aligned}&\mathsf{state}_{\mathcal{A}}(A,B,g)\\ &=\![X]\!\Rightarrow\\ &\mathsf{state}_{\mathcal{A}}(A,B,g,\exp(g,X),X).\mathsf{iknows}(\exp(g,X))\end{aligned}$$

$$\begin{aligned}&\mathsf{state}_{\mathcal{A}}(A,B,g,\exp(g,X),X).\mathsf{iknows}(\exp(g,Y))\\ &=\![Msg]\!\Rightarrow\\ &\mathsf{state}_{\mathcal{A}}(\ldots).\mathsf{iknows}(\{\!|A,Msg|\!\}_{\exp(\exp(g,X),Y)})\end{aligned}$$

□

This demonstrates the weak points of this naive schema (that reflects the state-of-the art in the previous Alice-and-Bob-style languages): in the second transition, A will accept only messages of the form $\exp(g,Y)$, while in reality, nobody can check this for an unknown Y. In fact, A should accept any incoming message GY here and build the Diffie-Hellman key for the outgoing message as $\exp(GY,X)$. We now sketch how such a translation is computed in general, in particular the appropriate check of incoming messages and the correct construction of outgoing messages.

Two Views The above example shows that agents are often unable to check that messages have exactly the structure that the protocol demands. To reason about this correctly, our translation from AnB to IF represents all messages from two kinds of *views* m^d, where m is the format that the message is supposed to

[8] There is nothing that forces us to execute such a rule when it is enabled.

have according to the protocol (the first view) and d is what is visible to the agent in question (the second view). For the second view, we introduce a new set of variables $\mathcal{X}_1, \mathcal{X}_2, \ldots$; these variables are used to label (parts of) messages of which the agent cannot see the structure. For instance, when in our running example A receives the second message from B, her knowledge looks like this:

$$A^{\mathcal{X}_1}, B^{\mathcal{X}_2}, g^{\mathcal{X}_3}, \exp(g, X)^{\exp(\mathcal{X}_3, \mathcal{X}_4)}, X^{\mathcal{X}_4}, \exp(g, Y)^{\mathcal{X}_5} \quad .$$

A can see the structure of the half key $\exp(g, X)$ because she has constructed it herself, while she cannot see the same for $\exp(g, Y)$; here her view is just a variable $\mathcal{X}_5$. We define appropriate deduction rules on such labeled terms so that, in our example, A can generate the full key $\exp(\exp(g, X), Y)$ as follows:

$$\frac{\dfrac{\overline{\exp(g, Y)^{\mathcal{X}_5}} \quad \overline{X^{\mathcal{X}_4}}}{\exp(\exp(g, Y), X)^{\exp(\mathcal{X}_5, \mathcal{X}_4)}}}{\exp(\exp(g, X), Y)^{\exp(\mathcal{X}_5, \mathcal{X}_4)}}$$

The root label $\exp(\mathcal{X}_5, \mathcal{X}_4)$ exactly describes how A obtains the key from the components of her knowledge.

More generally, whenever an agent should generate an outgoing message m according to the protocol, the AnB translator checks that m^d can be deduced from the agent's knowledge at that point for some *derivation* d. Then, d is the term of the outgoing message in the IF transition rule that the translator produces.[9] If there is no such derivation, then the translation rejects the protocol as "not executable": the agent cannot generate the outgoing message from its knowledge, which means that there is an error in the AnB specification.

Checking Messages We now define how agents can check the messages they receive. The idea is to ask whether an agent can derive from its knowledge, in two distinct ways, two terms that are supposed to be the same according to the protocol, i.e. deduce m^{d_1} and m^{d_2} with $d_1 \not\approx d_2$. Each such pair d_1 and d_2 of derivations represents a possible check that the agent can perform on messages, namely constructing the derivations and checking that the results are indeed the same. An example is that an agent receives a hash value $h(m)^{\mathcal{X}_1}$ in some message where it does not know the hashed message m and thus cannot check the structure. Now say that, in a later transition, the agent receives $m^{\mathcal{X}_2}$. He can then check that $\mathcal{X}_1 \approx h(\mathcal{X}_2)$, i.e. that applying the hash function h to the message $\mathcal{X}_2$ gives the same as $\mathcal{X}_1$.

More generally, we integrate such checks into the transition rules of honest agents, e.g. in the hash example we have a transition rule like this:

$$\mathsf{state}_{\mathcal{R}}(\mathcal{X}_1, \ldots).\mathsf{iknows}(\mathcal{X}_2) \mid \mathcal{X}_1 = h(\mathcal{X}_2) \Rightarrow \ldots$$

[9] Indeed, there are usually several different derivations for the same message m; the semantics, however, ensures that they are all equivalent derivations, so the choice of the derivation does not influence the rule meaning as explained in [56].

There are in general infinitely many such possible checks; for instance, in the above example, one may similarly check $h(\mathcal{X}_1) \approx h(h(\mathcal{X}_2))$ and so forth. Moreover, the message deduction problem is in general undecidable. Still, [56] shows that for an example theory that includes exponentiation, exclusive or, and other standard operators, we can decide message deduction and compute a finite sufficient set of checks for a given knowledge. Sufficient here means that the same set of incoming messages are accepted with the reduced finite set of checks. For a more detailed discussion on what an agent can recognize, such as the correct decryption of messages, we refer the reader to [56].

To summarize, the rules for A in our running example look as follows, where, for readability, we use instead of $\mathcal{X}_i$ the more intuitive variables from the AnB specification and replace constants wherever possible:

$$\begin{array}{l}\mathsf{state}_{\mathcal{A}}(A, B, g)\\ =[X]\Rightarrow\\ \mathsf{state}_{\mathcal{A}}(A, B, g, \exp(g, X), X).\mathsf{iknows}(\exp(g, X))\end{array}$$

$$\begin{array}{l}\mathsf{state}_{\mathcal{A}}(A, B, g, \exp(g, X), X).\mathsf{iknows}(GY)\\ =[Msg]\Rightarrow\\ \mathsf{state}_{\mathcal{A}}(\ldots).\mathsf{iknows}(\{\!|A, Msg|\!\}_{\exp(GY,X)})\end{array}$$

2.4 Channels

Much effort has been recently devoted to the composition of protocols, e.g. [3,26,30,31,40,41] to name a few works. We often have vertical composition of protocols, i.e. one protocol is run on top of another. For instance, we may have a banking service that runs over a "secure channel" that is provided by another protocol such as TLS or the authenticated Diffie-Hellman key exchange of our running example. It is desirable not to verify the entire composed system as a whole, but to verify the components individually, for instance, verify that TLS indeed provides a secure channel and that the banking service satisfies its goals when run over a secure channel. This compositional reasoning approach has several advantages over the monolithic verification approach. First, the smaller system components are usually easier to verify. Second, we have a greater reusability of verification results, as we can exchange TLS with any other protocol that provides a secure channel without repeating the analysis of the application protocol. Similarly, we can use TLS for other applications that rely on a secure channel without repeating the analysis of TLS anymore.

Thus, the notion of channel can provide a useful interface for a modular presentation and compositional verification of protocols and services. To that end, we need to define what it means that a protocol *provides* a channel with particular properties as a goal, and what the assumption of such a channel in a protocol means.

Channels as Assumptions We now sketch our model of channels as assumptions. Actually, we give two models, an abstract and a more concrete one, and

then prove them equivalent, so that we can use them interchangeably and the analysis methods can pick the one that suits them best.

The first model is called the *cryptographic channel model CCM*. The idea is that we can realize the channel properties by cryptographic means. We just give the example of authentic channels: we encode an authentic message M from A to B as $\{\mathsf{atag}, B, M\}_{\mathsf{inv}(ak(A))}$. Here, $\{\cdot\}_{\cdot}$ denotes asymmetric encryption, $(ak(A), \mathsf{inv}(ak(A)))$ is a dedicated public/private-key pair for realizing authentic channels, and atag is a special tag to distinguish this encoding from other digital signatures. $ak(A)$ is public for every agent A, and the intruder initially knows $\mathsf{inv}(ak(A))$ of every dishonest agent A.

The name of the intended recipient is included in the signed part of the message and is thus part of the authenticated information. This does not prevent other agents from reading the message, but makes clear for whom the message is meant.[10] This is just one of many possible ways to ensure authentic channels; one may similarly use MACs, for instance. The encoding of other channels similarly uses cryptography to ensure the channel properties.

To illustrate this encoding, the rules of A in the running example now become:

$$\begin{array}{l}\mathsf{state}_{\mathcal{A}}(A, B, ak(A), \mathsf{inv}(ak(A)), ak(B), g)\\ =[X]\Rightarrow\\ \mathsf{state}_{\mathcal{A}}(A, B, ak(A), \mathsf{inv}(ak(A)), ak(B), g, \exp(g, X), X).\\ \mathsf{iknows}(\{\mathsf{atag}, B, \exp(g, X)\}_{\mathsf{inv}(ak(A))})\end{array}$$

$$\begin{array}{l}\mathsf{state}_{\mathcal{A}}(A, B, ak(A), \mathsf{inv}(ak(A)), ak(B), g, \exp(g, X), X).\\ \mathsf{iknows}(\{\mathsf{atag}, A, GY\}_{\mathsf{inv}(ak(B))})\\ =[Msg]\Rightarrow\\ \mathsf{state}_{\mathcal{A}}(\ldots).\mathsf{iknows}(\{|A, Msg|\}_{\exp(GY,X)})\end{array}$$

We now present the second model of channels as assumptions, the *Ideal Channel Model ICM*, in which we use special persistent fact symbols for messages on different kinds of channels. The rules of A in our running example, for instance, are expressed as follows:

$$\begin{array}{l}\mathsf{state}_{\mathcal{A}}(A, B, g)\\ =[X]\Rightarrow\\ \mathsf{state}_{\mathcal{A}}(A, B, g, \exp(g, X), X).\mathsf{athCh}_{A,B}(\exp(g, X))\end{array}$$

$$\begin{array}{l}\mathsf{state}_{\mathcal{A}}(A, B, g, \exp(g, X), X).\mathsf{athCh}_{B,A}(GY)\\ =[Msg]\Rightarrow\\ \mathsf{state}_{\mathcal{A}}(\ldots).\mathsf{iknows}(\{|A, Msg|\}_{\exp(GY,X)})\end{array}$$

$\mathsf{athCh}_{A,B}(M)$ represents a message M that was sent by agent A on an authentic channel and meant for agent B. The behavior of the channels with respect to

[10] We need this to ensure the correspondence with the standard definition of authentic channels where it counts as an attack if b thinks that a message was sent by a to b while a actually meant to send it to somebody else.

```
Protocol : Authentic channel
Types :
  Agent A′, B′;
  Number Msg′;
  Function pk;
Knowledge :
  A′ :  A′, B′, pk(A′), inv(pk(A′));
  B′ :  B′, pk(A′);
Actions :
  A′ → B′ : {B′, Msg′}_inv(pk(A′))
Goals :
  A′ •→ B′ : Msg′
```

Fig. 2. AnB specification of a protocol realizing an authentic channel.

the intruder is then defined by rules such as:

$$\mathsf{iknows}(B).\mathsf{iknows}(M).\mathsf{dishonest}(A) \Rightarrow \mathsf{athCh}_{A,B}(M)$$
$$\mathsf{athCh}_{A,B}(M) \Rightarrow \mathsf{iknows}(M)$$

The first rule expresses that the intruder can send messages on an authentic channel to any agent B but only under the name of a dishonest agent A.[11] The second rule expresses that the intruder can receive any message on an authentic channel. There are similar rules for the abilities of the intruder on the other kinds of channels.

As shown in [57], the two models are equivalent (under certain conditions). Thus, the CCM is a correct realization of the ICM and we can use both models interchangeably.

Channels as Goals The goals of a protocol can be specified using the different kinds of channels as in our running example where we have specified the secure transmission of a payload message *Msg* as a goal. These goal definitions are close to standard ones of security protocols, e.g. [7,48,54]. For authentication goals, we use auxiliary events, as we have done before for secrecy goals. This allows us to express goals in a protocol-independent way.

Compositionality The study of compositionality with respect to channels has revealed several subtle details about what must be required of a channel (implying, for instance, the inclusion of the intended recipient on an authentic channel).

[11] The intruder knows all agent names by assumption, but we need $\mathsf{iknows}(B)$ on the left-hand side because IF requires all variables that appear on the right-hand side to be already present on the left.

`Protocol` : *Authenticated Diffie-Hellman key exchange, version 2*

`Types` :

`Agent` A, B;

`Number` g, X, Y, Msg;

`Function` pk;

`Knowledge` :

A : $A, B, g, pk(A), \mathsf{inv}(pk(A))$;

B : $B, g, pk(A)$;

`Actions` :

$A \rightarrow B : \{B, \exp(g, X)\}_{\mathsf{inv}(pk(A))}$

$B \bullet\!\!\rightarrow A : \exp(g, Y)$

$A \rightarrow B : \{\!|A, Msg|\!\}_{\exp(\exp(g,X),Y)}$

`Goals` :

$A \bullet\!\!\rightarrow\!\!\bullet B : Msg$

Fig. 3. AnB specification of the authenticated Diffie-Hellman key exchange, composed with the realization of an authentic channel.

As an example, consider the protocol in Fig. 2 as a way to realize an authentic channel, as it is assumed in our running example of the authenticated Diffie-Hellman key exchange; $pk(A')$ and $\mathsf{inv}(pk(A'))$ are the public and the private key of A', respectively. We can thus, for instance, implement the first authentic channel of the running example by the protocol of Fig. 2 to obtain the protocol shown in Fig. 3.

In [57], we give suitable definitions and conditions to obtain the desired compositionality results, namely that an assumed channel can be realized by any protocol that provides it as a goal.

Pseudonymous Channels We have generalized all the above models and results to include channels where agents may alternatively be identified by pseudonyms rather than by their real names. Pseudonymous channels are created by techniques like purpose-built keys (PBK) or TLS without client authentication: we have something similar to a secure channel except that one end is not identified by its real name but by some pseudonym, which is usually related to an unauthenticated public-key; see, e.g., [19,33,42,47]. In the case of authentic channels, this concept has often been referred to as *sender invariance*: the receiver can be sure that several messages come from the same source, whose real identity is not known or not guaranteed. However, there is more to it.

First, pseudonymous channels, both as assumptions and as goals, should not be defined as entirely new concepts unrelated to the previous channels. Rather,

`Protocol` : *Diffie-Hellman key exchange without client authentication*

`Types` :

`Agent` A, B;

`Number` g, X, Y, Msg;

`Knowledge` :

A : A, B, g;

B : B, g;

`Actions` :

$A \rightarrow B : \exp(g, X)$

$B \bullet\!\!\rightarrow A : \exp(g, Y)$

$A \rightarrow B : \{\!|A, Msg|\!\}_{\exp(\exp(g,X),Y)}$

`Goals` :

$[A] \bullet\!\!\rightarrow\!\!\bullet B : Msg$

Fig. 4. AnB specification of the Diffie-Hellman key exchange without client authentication.

we define them as variants of the standard channels discussed above where one (or both) ends are identified by a pseudonym rather than the real name.[12]

Second, the concept of pseudonymous channels is useful to model a number of scenarios. The most common one is probably the above mentioned TLS without client authentication as it is common in the Internet: it is in a sense weaker than a standard secure channel, but (assuming the server's public key is properly authenticated) it is sufficient for submitting a client's password over this channel to achieve full authentication. We thus want to use such a channel both as a goal for protocols like TLS where only one side is authenticated, and as an assumption in high-level protocols that use such a channel for a login, for instance.

In AnB, we write $[A]_\psi$ to denote the identity of an agent A that is not identified by its real name A but by some pseudonym ψ, e.g. we write $[A]_\psi \bullet\!\!\rightarrow B : M$ for an authentic channel. We also allow that the specification of ψ is omitted, and write only $[A] \bullet\!\!\rightarrow B$, when the role uses only one pseudonym in the entire session (which is the case for most protocols). The omitted variant is a short-cut for a pseudonym that A freshly generates when it first uses a pseudonymous channel.

Example 2. The protocol in Fig. 4 establishes a secure channel between an unauthenticated A, which uses its Diffie-Hellman half key $\exp(g, X)$ as a pseudonym, and an authenticated B (just as in the case of TLS). Such a channel is good enough for a login protocol in which a client A transmits her user name and password, and thereby authenticates herself to a server B. □

[12] One may even argue that real names are also just a kind of pseudonym, so there is no difference at all. In our model, the difference between real names and pseudonyms is that we assume that real names uniquely identify agents and do not change over time, while pseudonyms may be arbitrarily created by any agent. As a consequence, every agent (including the intruder) can act under several identities.

3 Constraint-Based Model Checking

In the previous section, we have discussed the modeling of protocols and their properties, and thereby set up a challenging task for automated verification. We now discuss how OFMC addresses this challenge.

3.1 The Lazy Intruder

The naive exploration of the search space generated by such a specification including a Dolev-Yao-style intruder is not feasible, due to a large or even infinite number of messages that the intruder can construct and send from a given set of known messages. One of the core ideas in OFMC (and, similarly, in several other approaches, e.g. [1,2,17,18,22,23,25,37,43,53]) is to avoid this naive enumeration by using a symbolic, constraint-based approach, which allows us to significantly reduce the search space without excluding attacks (and without introducing new ones).

Let us illustrate this with an example. Assume that in order to carry out an attack, the intruder needs to send to an honest agent a a message that has the form $\{N\}_{pk(a)}$ for some number N encrypted with a's public key $pk(a)$. This can be satisfied in several different ways. First, the intruder can take any term t that he knows and encrypt it with a's public key and send $\{t\}_{pk(a)}$. Alternatively, he can send instead any message of the form $\{\cdot\}_{pk(a)}$ that he knows (even though he cannot decrypt it). Using constraints, however, we do not explore all these possibilities directly, but rather work with the symbolic term $\{N\}_{pk(a)}$ (i.e. leaving the variable N) and impose the constraint $\mathit{from}(\{\,\{N\}_{pk(a)}\,\}; IK)$ where IK is the set of messages known to the intruder at this point (the current *intruder knowledge*). This constraint means that, whatever N is, the intruder must be able to construct the term $\{N\}_{pk(a)}$ from the knowledge IK. We thus base the protocol analysis on a *constraint satisfaction problem*. This is done in a demand-driven way, i.e. we postpone the substitution of variables as long as possible during search. For this reason, we call the technique the *lazy intruder*.

In general, a constraint has the form

$$\mathit{from}(T; IK)$$

where T and IK are both sets of message terms with variables. The models of such a constraint are those interpretations $\mathcal{I}$ of the variables such that $T^{\mathcal{I}}$ can be generated from $IK^{\mathcal{I}}$ using the rules of the intruder to construct and deconstruct messages.

The lazy intruder technique uses the notion of *simple* constraint, i.e. a constraint in which all terms to be generated are variables. This simple form is always satisfiable as the intruder can always generate some message. The idea is to reduce a given constraint to an equivalent set of simple constraints. Here, "equivalent" refers to the set of models of a constraint, i.e. the satisfying interpretations of the variables. We thus formulate, for a given intruder model, a set of constraint reduction rules that are *correct* (in the sense that they maintain

the set of models) and *terminating* (meaning that we arrive, after finitely many steps, at a finite set of simple constraint sets).

The constraint reduction rules of the lazy intruder are of three kinds:

- *generation* rules that describe how the intruder can compose messages from known ones,
- *analysis* rules that describe how he can decompose messages,
- and finally there is a *unification* rule that expresses that the intruder can use unifiable messages from his knowledge to fulfill the constraint.

A set of such rules is given in [12,58], where all the details of the formal constraint reduction are spelled out. Here, we focus only on the main ideas by means of an example.

Example 3. To understand the way the lazy intruder works, let us now consider the non-authenticated Diffie-Hellman key exchange, where the half keys are sent on insecure channels. Consider the following abstract execution trace, where we first ignore the question of whether the intruder can generate any acceptable message and we just use variables for messages sent by the intruder:

$$\begin{array}{l} 1.\ a \to i(b) : \exp(g, x) \\ 2.\ i(b) \to a : M_1 \\ 3.\ a \to i(b) : \{\!|msg|\!\}_{\exp(M_1,x)} \\ \qquad\qquad\quad \mathsf{secret}(msg, b) \end{array}$$

Here, the agent a sends her Diffie-Hellman half key $\exp(g, x)$ to an agent b, but the message is intercepted by the intruder, which we display as $i(b)$.[13] The intruder then replies by sending some message M_1 that a parses as the Diffie-Hellman half key from b. She thus sends the payload message msg symmetrically encrypted with the resulting Diffie-Hellman full key $\exp(M_1, x)$. She also declares the payload as a secret with b. We want now to check whether the intruder can find out this secret, assuming the initial intruder knowledge $IK_0 = \{g\}$. This is formalized by the following constraint set:

$$\begin{array}{ll} \mathit{from}(\{M_1\}; IK_1) & \text{where } IK_1 = IK_0 \cup \{\exp(g, x)\} \\ \mathit{from}(\{msg\}; IK_2) & \text{where } IK_2 = IK_1 \cup \{\{\!|msg|\!\}_{\exp(M_1,x)}\} \end{array}$$

We will now describe only one sequence of reduction steps that leads to the solution of the constraint set, while the actual constraint reduction procedure considers also other reduction sequences, which in this case lead to a dead end (i.e. to unsatisfiable constraints). We follow the path that the intruder successfully decrypts the encrypted message in IK_2. This adds a new constraint that he can indeed generate the key $\exp(M_1, x)$ and we add the cleartext msg to the intruder knowledge IK_2 in the second constraint. The new constraint set after

[13] Actually, due to our identification of intruder and network for insecure channels, the intruder intercepts *every* message transmitted on such channels.

this step of the constraint reduction procedure looks a follows:

$$\begin{array}{l} \mathit{from}(\{M_1\}; IK_1) \\ \mathit{from}(\{\mathit{msg}\}; IK_2 \cup \{\mathit{msg}\}) \\ \mathit{from}(\{\exp(M_1, x)\}; IK_2) \end{array}$$

Obviously, the second constraint can now be solved (by applying the unification rule) and removed from the constraint set. We next turn to the key-generation constraint. Observe that the intruder cannot directly compose this message as he does not know a's secret value x, but there is a way to compose this term if M_1 has the form $\exp(M_2, M_3)$ for two new variables M_2 and M_3. The resulting form of the key $\exp(\exp(M_2, M_3), x)$ is equivalent to $\exp(\exp(M_2, x), M_3)$ according to the algebraic properties of exponentiation:

$$\begin{array}{l} \mathit{from}(\{\exp(M_2, M_3)\}; IK_1) \\ \mathit{from}(\{\exp(\exp(M_2, x), M_3)\}; IK_2) \end{array}$$

This latter representation can indeed be composed, i.e. by an application of a generation rule we get:

$$\begin{array}{l} \mathit{from}(\{\exp(M_2, M_3)\}; IK_1) \\ \mathit{from}(\{\exp(M_2, x), M_3\}; IK_2) \end{array}$$

Now we can unify the term $\exp(M_2, x)$ with a's half key, i.e. setting $M_2 = g$ we obtain:

$$\begin{array}{l} \mathit{from}(\{\exp(g, M_3)\}; IK_1) \\ \mathit{from}(\{M_3\}; IK_2) \end{array}$$

Finally, since IK_1 contains g we can generate the term in the first constraint, leaving a set of simple constraints:

$$\begin{array}{l} \mathit{from}(\{M_3\}; IK_1) \\ \mathit{from}(\{M_3\}; IK_2) \end{array}$$

Thus, the intruder can perform the attack for any value M_3 that he knows. □

3.2 Algebraic Reasoning

The above example briefly touched the subject of algebraic reasoning. Now we give an overview of what OFMC supports and how. In fact, as part of the parameters of OFMC, one can specify an algebraic theory that defines the (cryptographic) operators and their algebraic properties.

Finite Theories To begin with, we allow *finite theories*, i.e. theories under which every term has a finite equivalence class. The exponentiation property that $\exp(\exp(G, X), Y) \approx \exp(\exp(G, Y), X)$ is an example, because, intuitively, there are only finitely many re-arrangements of exponents in any term. Note that unification for finite theories is in general already an undecidable problem; thus,

we cannot handle such specifications without any restrictions. The approach of OFMC is to limit the number of instantiations of a variable in the form that we had in Example 3. The fact that we only bound the handling of variables (and not the terms that can be substituted) directly ensures that for many theories (like the exponentiation example) the restriction is without loss of generality, i.e. no solutions are excluded. In the example of exponentiation, we have to specify the following hint for OFMC in the theory file:[14]

```
topdec(exp,exp(T1,T2))=
  [T1,T2]
  if T1==exp(Z1,Z2){
    [exp(Z1,T2),Z2]}
```

This specifies the possible solutions of the unification problem $\exp(\cdot,\cdot) \approx \exp(T_1, T_2)$, i.e. the different ways to compose the term $\exp(T_1, T_2)$ using exp as a top-level symbol. (Composition with other symbols does not yield an exponentiation-term in our algebraic theory.) The first solution is the standard "syntactical" solution, i.e. an exponentiation of the subterms; this is possible for any operator. Second, if (recursively) T_1 can be unified $\exp(Z_1, Z_2)$, then there is also an alternative composition using subterms $\exp(Z_1, T_2)$ and Z_2; this is exactly the case used in the above example. Note that this describes a recursive procedure, as composition/decomposition of the term T_1 may give rise to further such checks for exp-decompositions. In the case that the given term is a variable, this recursion can be repeated arbitrarily, but we bound this in OFMC, however, for the exponentiation case, this bounding is without loss of solutions. More generally, the `topdec`-specifications, like the one above, give a skeleton for a unification algorithm modulo the described theory. Namely, a recursive structure to find all solutions, and one can either bound the instantiation (sacrificing completeness in general) or leave it unbounded (sacrificing termination).

Cancellation Theories Many algebraic properties are of the form $\{|\{|m|\}_k|\}_k \approx m$, which intuitively expresses that "decryption and encryption cancel each other out". The characteristic is that the right-hand side of each equation is either a subterm of the left-hand side or a constant. This underlying orientation of the rules gives rise to a system of rewrite rules that simplifies given message terms. We interpret these rewrite rules modulo the finite theory we have considered before and require that the resulting rewrite relation is convergent and terminating, so that every term has a unique normal form (modulo the finite theory). When using symbolic terms, however, this implies several potential termination problems. As before, to enforce termination, we consider here a bound on the instantiation of variables, which in many cases is not a restriction. The key idea is to analyze the given intruder knowledge as far as possible using cancellations.

[14] OFMC requires that the specification of the algebraic theory contains such hints in order to guide the analysis procedure. A similar requirement holds also for the cancellation theories discussed below.

Formally, we say that the intruder knowledge is *completely analyzed* iff the normal form of every deducible term is either contained in the intruder knowledge, or can be composed from it (without cancellation). Thus, in a completely analyzed knowledge, we do not need to consider any more analysis steps or cancellation properties, but only composition and unification. In the case of the lazy intruder, due to the variables in the intruder knowledge, this notion is always related to a set of constraints on these variables.

Example 4. As an example, consider IK_2 from Example 3, which is related to the constraint $from(M_1; IK_1)$ due to the variable M_1 in IK_2. This knowledge is not completely analyzed, because the key-term for the encrypted message is derivable, so the intruder can compose the term $\{|\{|msg|\}_{\exp(M_1,x)}|\}_{\exp(M_1,x)}$ and thus obtain msg, which cannot be obtained by composition alone (without the cancellation property). After the addition of msg to IK_2, it is completely analyzed. □

3.3 Symbolic sessions

Most protocol analysis tools allow the user only to specify a concrete analysis scenario consisting of different protocol sessions executed in parallel, such as "Alice wants to talk to Bob and in parallel to the intruder" (of course, Alice does not know which communication partners are honest and which are not). Such a manual specification is, however, cumbersome, especially since the number of scenarios to analyze for a given number of sessions grows exponentially.

OFMC, in contrast, also allows the user to simply specify the number of sessions that the user wishes to analyze, and covers *all* instantiations of the agents in these sessions. This is not only more convenient but also more efficient, since the enumeration of all scenarios is completely avoided. The trick is to use the lazy intruder technique to instantiate agent names whenever necessary. More precisely, we consider an initial state with variables for all agent names (possibly with constraints like $A \neq B$ or $A \neq i$). The lazy intruder starts with the constraint $from(\{A_1, \ldots, A_n\}; IK_0)$ where A_i are agent names of the initial state and IK_0 is the initial intruder knowledge. We assume that all agent names are public knowledge that the intruder initially knows.[15] This approach reflects that "the intruder chooses the sessions" according to what may help him perform an attack, so instantiation is integrated into the protocol analysis. Since this is done lazily, the names are instantiated only when this matters for an attack. In general, we get attacks with variables for agent names; these attacks thus work for arbitrary agents.

3.4 Constraint Differentiation

The lazy intruder drastically reduces the size of the search tree generated during protocol analysis, providing an effective solution to the problem of the prolific

[15] We specify this by a dedicated intruder rule, so we do not need to enumerate this set in the intruder knowledge and can even consider an unbounded number of agents.

intruder, who can compose and send messages at will. However, the lazy intruder does not address the problem resulting from the large number of interleavings possible due to parallel protocol executions. In standard model-checking approaches for concurrent systems, the interleaving problem is often handled using *partial-order reduction* (*POR*), a technique that reduces the number of interleavings that need to be considered by exploiting independencies between the possible transitions [60]. One might expect that the lazy intruder could be directly combined with partial-order reduction. However, this combination is not effective as the different transitions of the lazy intruder rarely lead to the same (symbolic) successor state and therefore there is practically no independence of transitions that can be exploited by POR.

The *constraint differentiation* technique [58] effectively integrates the lazy intruder and ideas from POR by using independence information from the symbolic transition system when reducing constraints. Constraint differentiation works by introducing a new kind of constraint. However, existing constraint-based methods for the various symbolic intruder approaches [1,2,17,18,22,23,25,37,43,53] do not need to be individually updated for constraint differentiation since we have defined our technique in a generic way, namely as a transformation to "differentiate" a given symbolic intruder approach (that already is correct and terminating for a particular intruder model).

We again make use of an example to illustrate the main ideas. Assume that when the agent a receives a message of the form $\{\!|a, X|\!\}_k$, it replies with $\{\!|X|\!\}_{k'}$, for a key k'. Let us call this transition θ_1. Assume further that there is another agent b waiting for a message of the form $\{\!|b, Z|\!\}_{k'}$ to which it replies with $\{\!|Z|\!\}_{k'}$. Let us call this transition θ_2. These two transitions can be performed in either order, although they will produce different constraints: if θ_1 is followed by θ_2, then we have the constraints

$$\begin{array}{l} \mathit{from}(\{\ \{\!|a, X|\!\}_k\ \}; \mathit{IK}) \\ \mathit{from}(\{\ \{\!|b, Z|\!\}_{k'}\ \}; \mathit{IK} \cup \{\ \{\!|X|\!\}_{k'}\ \}) \end{array}$$

while if θ_2 is followed by θ_1, then we have

$$\begin{array}{l} \mathit{from}(\{\ \{\!|b, Z|\!\}_{k'}\ \}; \mathit{IK}) \\ \mathit{from}(\{\ \{\!|a, X|\!\}_k\ \}; \mathit{IK} \cup \{\ \{\!|Z|\!\}_{k'}\ \}) \end{array}$$

These two sets of constraints represent overlapping but different sets of solutions, due to the different intruder knowledges. The main idea of constraint differentiation is to exploit this overlap by restricting the solutions of one of the constraint sets, say the first one, to the solutions not covered by other one. To that end, we introduce a new kind of constraint that is capable of expressing this difference. For instance, the constraints of the execution "θ_1 followed by θ_2" may in some cases be replaced by the following constraints:

$$\begin{array}{l} \mathit{from}(\{\ \{\!|a, X|\!\}_k\ \}; \mathit{IK}) \\ \mathit{D\text{-}from}(\{\ \{\!|b, Z|\!\}_{k'}\ \}; \mathit{IK}; \{\ \{\!|X|\!\}_{k'}\ \}) \end{array}$$

More generally, we introduce *D-from* constraints of the form

$$\mathit{D\text{-}from}(T; \mathit{IK}; \mathit{NIK})\,,$$

where, intuitively, *NIK* represents new messages that are not in IK; the acronym stands for *new intruder knowledge*. The constraint formalizes that the set of terms T must be generated by the intruder using the knowledge in the set $IK \cup NIK$, but we are only interested in solutions that employ new information in *NIK* and hence we exclude all solutions of $\mathit{from}(T; IK)$.

Assume now that the key k' cannot be generated from IK. Then the only way to satisfy the *D-from* constraint is to unify $\{|X|\}_{k'}$ and $\{|b, Z|\}_{k'}$, i.e. $X \mapsto b, Z$. In particular, the *D-from* constraint forbids using other messages encrypted with k' that occur in IK, if any.

In combination with other constraints, this can rule out the entire interleaving. For instance, if $\{|a, n|\}_k \in IK$ and from IK we cannot derive any other message encrypted with k, then the only solution allowed by the first constraint is $X = n$. Therefore X and b, Z do not unify and the resulting constraints are unsatisfiable. This shows how constraint differentiation can either limit the possible solutions for one execution order or even rule out a particular execution order.

4 The Fixed-Point Module

In recent years, several techniques and tools have been developed that address the problem of protocol verification with an unbounded number of sessions by employing over-approximation techniques, e.g. [13,15,16,38,39]. Over-approximation means that one considers a model that allows strictly more traces or reachable states than the original model. This can induce attacks that are *false positives*, i.e. attacks that work only in the over-approximated model, but not in the original model. 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, if successful, implies that the original model is safe as well.

We have implemented a prototype of a new module for OFMC that is based on such over-approximation ideas. The result of a verification in this module is a fixed-point of facts that can ever occur in any reachable state, and we thus call it the *fixed-point module*. This module complements the "classical" OFMC: we analyze a protocol first in the classical setting with a bounded number of sessions which may yield an attack. Otherwise, if the protocol is safe for a given number of sessions, then we complete the verification by using the new fixed-point module.

Due to the subtlety of protocol models, it is often not immediate that the considered model is actually an over-approximation of the original model. This is, however, the crucial assumption of the entire approach. Therefore, we have investigated the relationships between several such models in [55]. There, we have shown that for a large class of protocols we indeed have an over-approximation relationship which allows us to conclude that the approach behind the fixed-point module is indeed sound.

We now discuss in more detail two kinds of over-approximation that we use in OFMC's fixed-point module.

4.1 Data Abstraction

A common form of abstraction in many verification tools (not only in protocol verification) is based on the idea of abstract interpretation [28]. We refer to this as *data abstraction*, because we map the infinite set of (fresh) data to finitely many equivalence classes; we then consider the abstract equivalence classes instead of the concrete data.

For instance, in our running example, we can abstract all the exponents that an honest agent a freshly generates in all sessions of the protocol into one that we denote by $\mathit{exponent}(a)$. As a result, if the intruder cannot manipulate any of the half keys, two honest agents a and b will end up with the same key $\exp(\exp(g, \mathit{exponent}(a)), \mathit{exponent}(b))$ in every session.

This technique has an important prerequisite: there may not be any negative comparisons in the rules, e.g. that two half keys must be different, or else the abstract model would not be an over-approximation of the concrete one. This limits the class of protocols that can be handled with such methods, but we can easily check that a given specification meets such conditions.

4.2 Control Abstraction

We also consider another form of over-approximation that was first considered in planning [14] and that makes sense in combination with data abstraction and its assumptions: *control abstraction*. The idea here is that the order in which certain facts are "reached" does often not matter, and we can rather just consider what facts occur in any reachable state, in particular which messages the intruder can ever find out.

The fixed-point that we obtain for the running example under control abstraction represents a situation where the intruder has obtained the half key $\exp(g, \mathit{exponent}(a))$ of each agent a and the exponent $\mathit{exponent}(a)$ of each dishonest agent a. Moreover, he has not obtained the full key $\exp(\exp(g, \mathit{exponent}(a)), \mathit{exponent}(b))$ of any pair of honest agents a and b. Finally, in every local state of an honest agent a that has negotiated a full key for communication with another honest agent b, this full key is $\exp(\exp(g, \mathit{exponent}(a)), \mathit{exponent}(b))$.

This concludes our brief exposition of the fixed-point module of OFMC and we now consider some experimental results.

5 Experimental Results

We have applied OFMC to a large number of industrial-strength protocols including all the protocols in the AVISPA Library [9], which contains about 70 real-world protocols such as SET, IKE v.2, Kerberos in different variants, TLS,

and H.530. Detailed experiments with running times and comparisons with other tools can be found in [8,12,54,58].

As a concrete example, we summarize here our analysis of the H.530 protocol [44], a protocol developed by Siemens and proposed as an Internet standard for multimedia communications. We have modeled the protocol in its full complexity and OFMC detected a replay attack serious enough that Siemens revised the protocol [45].

5.1 The H.530 Protocol

The H.530 protocol [44,45] provides mutual authentication and key agreement in mobile roaming scenarios in multimedia communication. H.530 is deployed as follows: a mobile terminal (MT) wants to establish a secure connection and negotiate a Diffie-Hellman key with the gatekeeper (VGK) of a visited domain. As they do not know each other in advance, the authentication is performed using a trusted authentication facility AuF within the home domain of the MT.

Fig. 5 shows the message exchange of H.530 in AnB notation (slightly simplified). There is initially a shared key between the mobile terminal MT and its home server AuF, denoted by $sk(MT, AuF)$, as well as a shared key between the visited gatekeeper VGK and AuF, denoted by $sk(VGK, AuF)$. In the first message, MT sends out a request that contains a fresh Diffie-Hellman half key $\exp(g, X)$. This message, like all the following ones, is "MACed" (Message Authentication Code): a hash value of the message using a *keyed* hash function is added to the message. A keyed hash function is like a normal hash function, but has as an extra parameter a symmetric key, so that only participants who know the key can construct—or check—the hash value.[16]

Since the first message from MT is MACed using the key $sk(MT, AuF)$, the receiver VGK can read the Diffie-Hellman half key and the name of MT (at least what it seems to be), but cannot check the authenticity of the messages. In the second message, VGK forwards this request to AuF, including his own fresh Diffie-Hellman half key $\exp(g, Y)$ which is XOR-ed to the half key from MT.

After having checked that all MACs are "adding up", the AuF answers in the third message with an acknowledgment, which contains both half keys and the name of the participants. As in message 2, we have two nested MACs, the outer one with $sk(VGK, AuF)$ and the inner one with $sk(MT, AuF)$. Observe, however, that this time the inner one is without a copy of the cleartext. That is exactly the weakness of the protocol that we will describe below. The last two messages between MT and VGK are MACed using the new Diffie-Hellman key of MT and VGK, proving that both can construct the key. Note that MT

[16] Our model of a MAC is based on using a simple implementation using an unkeyed hash function: the key is concatenated together with the message to hash. We do not need to discuss the cryptographic requirements and implications of such an implementation, since in our model the MAC has exactly the properties we want: one can build a MAC iff one knows the key and the MACed message, and one cannot recover from a MAC the MACed message (even if one knows the key).

1. $MT \to VGK:\ req(MT, VGK, AuF, \exp(g, X))$
2. $VGK \to AuF: mac(sk(VGK, AuF), (req(MT, VGK, \exp(g, X)), VGK, ver(X, Y)))$
3. $AuF \to VGK: mac(sk(VGK, AuF), (VGK, ack(MT, VGK, AuF, X, Y)))$
4. $VGK \to MT:\ mac(dhk(X, Y), (ack(MT, VGK, AuF, X, Y), \exp(g, Y)))$
5. $MT \to VGK:\ mac(dhk(X, Y), (MT, VGK))$

where
$$mac(K, M) = (M, f(K, M))$$
$$dhk(X, Y) = \exp(\exp(g, X), Y)$$
$$ver(X, Y) = xor(\exp(g, X), \exp(g, Y))$$
$$req(MT, VGK, AuF, X) = mac(sk(MT, AuF), (MT, VGK, \exp(g, X)))$$
$$ack(MT, VGK, AuF, X, Y) = f(sk(MT, AuF), (VGK, ver(X, Y)))$$

Fig. 5. The message exchange of H.530 in AnB notation (slightly simplified).

[Normal session of the protocol (recorded by the intruder)]

$1'.\ i(mt) \to vgk:\ mt, vgk, auf, \exp(g, z), rand$
$2'.\ vgk \to i(auf): mac(sk(vgk, auf), (mt, vgk, auf, \exp(g, z), rand, vgk, ver(z, \exp(g, y_2))))$
$3'.\ i(auf) \to vgk: mac(sk(vgk, auf), (vgk, ack(mt, vgk, auf, x, y)))$
$4'.\ vgk \to i(mt):\ mac(dhk(z, y2), (ack(mt, vgk, auf, x, y), \exp(g, y2)))$
$5'.\ i(mt) \to vgk:\ mac(dhk(z, y2), (mt, vgk))$

Fig. 6. An attack on H.530, where *rand* and z are random values created by the intruder, and $y2$ is value created by vgk for Y in the second run of the protocol.

receives the half key from VGK also in cleartext, so that he can build the key to check the hash value used here.

The specification in OFMC took one work-day[17] and after an analysis time of 1.3 seconds, OFMC reported a replay attack, displayed in Fig. 6, which works as follows. First the intruder listens to a session between a set of honest agents mt, vgk, and auf. He uses the recorded messages later to "steal" the identity of mt, i.e. to pose as mt towards vgk and negotiate a new Diffie-Hellman key with it. More in detail, in message $1'$, vgk can only see the cleartext part of the message, the intruder can thus insert anything for the keyed hash (denoted by *rand* here). Note that the intruder creates a fresh Diffie-Hellman secret z here. The intruder intercepts message $2'$ from vgk to auf and for the reply $3'$ from auf, he replays the message 3 from the previous session, which is based on the old Diffie-Hellman secrets x and y. Observe that vgk cannot detect this, because these old values are only contained in the ack part of the message, which is a keyed hash using $sk(mt, auf)$. Thus, from vgk's point of view, the auf has just acknowledged the key exchange with mt, namely using the Diffie-Hellman key $dhk(z, y2)$. This key is known to the intruder, since he created z himself and

[17] At the time, OFMC did not support algebraic properties, so a work-around for Diffie-Hellman had to be implemented and XOR was replaced with concatenation.

can read $\exp(g, y2)$ for instance from the cleartext part of message $2'$. Therefore, he can complete the attack and *vgk* believes to share the new Diffie-Hellman key with *mt*. Due to this attack, Siemens has changed the protocol following our suggestion to include the Diffie-Hellman half keys in the MAC for *VGK* in message 3 [45].

6 Conclusions

We have presented the main features of the Open-source Fixed-point Model Checker, a state-of-the-art security protocol analysis tool. An ongoing line of work in the context of the AVANTSSAR project is concerned with extending the scope of OFMC towards the security analysis of service-oriented architectures. To that end, we are currently extending the compositional reasoning and abstraction techniques of OFMC.

Acknowledgments

The work presented in this paper was partially supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" and the PRIN'07 project "SOFT". We thank David Basin, Achim Brucker, Paul Hankes Drielsma, and all the members of the projects AVISS, AVISPA and AVANTSSAR for very fruitful discussions.

References

1. M. Abadi and V. Cortier. Deciding knowledge in security protocols under (many more) equational theories. In *Proceedings of CSFW'05*, pages 62–76. IEEE Computer Society Press, 2005.
2. R. Amadio and D. Lugiez. On the Reachability Problem in Cryptographic Protocols. In *Proceedings of CONCUR'00*, LNCS 1877, pages 380–394. Springer-Verlag, 2002.
3. S. Andova, C. Cremers, K. Gjøsteen, S. Mauw, S. Mjølsnes, and S. Radomirović. A framework for compositional verification of security protocols. *Information and Computation*, 206:425–459, 2008.
4. M. Arapinis, S. Delaune, and S. Kremer. From one session to many: Dynamic tags for security protocols. In *Proceedings of LPAR'08*, LNCS 5330, pages 128–142. Springer-Verlag, 2008.
5. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *Proceedings of CAV'05*, LNCS 3576, pages 281–285. Springer-Verlag, 2005.
6. A. Armando, R. Carbone, and L. Compagna. LTL Model Checking for Security Protocols. In *Proceedings of CSF'07*, pages 385–396. IEEE Computer Society Press, 2007.

7. AVISPA. Deliverable 2.3: The Intermediate Format. Available at `www.avispa-project.org`, 2003.
8. AVISPA. Deliverable 7.4: Assessment of the AVISPA Tool v.3. Available at `http://www.avispa-project.org`, 2005.
9. The AVISPA Library of Protocols. `http://www.avispa-project.org/library`.
10. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
11. D. Basin, S. Mödersheim, and L. Viganò. Algebraic intruder deductions. In *Proceedings of LPAR'05*, LNAI 3835, pages 549–564. Springer-Verlag, 2005.
12. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.
13. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of CSFW'01*, pages 82–96. IEEE Computer Society Press, 2001.
14. A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
15. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *Proceedings of CSFW'03*, pages 126–140. IEEE Computer Society Press, 2003.
16. Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Tree automata for detecting attacks on protocols with algebraic cryptographic primitives. In *Proceedings of the INFINITY'07 Workshop*, 2007 (to appear in ENTCS).
17. M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proceedings of ICALP'01*, LNCS 2076, pages 667–681. Springer-Verlag, 2001.
18. M. Boreale and M. G. Buscemi. A Framework for the Analysis of Security Protocols. In *Proceedings of CONCUR'02*, LNCS 2421, pages 483–498. Springer-Verlag, 2002.
19. S. Bradner, A. Mankin, and J. Schiller. A framework for purpose built keys (PBK), June 2003. Work in Progress (Internet Draft: `draft-bradner-pbk-frame-06.txt`).
20. F. Butler, I. Cervesato, A. Jaggard, and A. Scedrov. A formal analysis of some properties of Kerberos 5 using MSR. In *Proceedings of CSFW'02*, pages 175–190. IEEE Computer Society Press, 2002.
21. I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Relating Strands and Multiset Rewriting for Security Protocol Analysis. In *Proceedings of CSFW'00*, pages 35–51. IEEE Computer Society Press, 2000.
22. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proceedings of FST TCS'03*, LNCS 2914, pages 124–135. Springer-Verlag, 2003.
23. Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proceedings of ASE'01*, pages 373–376. IEEE Computer Society Press, 2001.
24. Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In *Proceedings of CAV'02*, LNCS 2404, pages 324–337. Springer-Verlag, 2002.
25. R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proceedings of SAS'02*, LNCS 2477, pages 326–341. Springer-Verlag, 2002.
26. V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, 2009.
27. V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

28. P. Cousot. Abstract interpretation. *Symposium on Models of Programming Languages and Computation, ACM Computing Surveys*, 28(2):324–328, 1996.
29. C. Cremers. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Proceedings of CAV'08*, LNCS 5123, pages 414–418. Springer-Verlag, 2008.
30. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 11–23. ACM Press, 2003.
31. S. Delaune, S. Kremer, and M. D. Ryan. Composition of password-based protocols. In *Proceedings of CSF'08*, pages 239–251. IEEE Computer Society Press, 2008.
32. G. Denker, J. K. Millen, and H. Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, 2000.
33. T. Dierks and C. Allen. RFC2246 – The TLS Protocol Version 1, Jan. 1999.
34. D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
35. 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.
36. S. Even and O. Goldreich. On the security of multi-party ping pong protocols. In *Proceedings of FOCS'83*, pages 34–39. IEEE Computer Society Press, 1983.
37. M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proceedings of CSFW'01*, pages 160–173. IEEE Computer Society Press, 2001.
38. T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proceedings of CADE'00*, LNCS 1831, pages 271–290. Springer-Verlag, 2000.
39. J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *Proceedings of FMPPTA'00*, LNCS 1800, pages 977–984. Springer-Verlag, 2000.
40. J. D. Guttman. Authentication tests and disjoint encryption: a design method for security protocols. *Journal of Computer Security*, 3–4(12):409–433, 2004.
41. J. D. Guttman. Cryptographic protocol composition via the authentication tests. In *Proceedings of FOSSACS'09*, LNCS 5504, pages 303–317. Springer-Verlag, 2009.
42. P. Hankes Drielsma, S. Mödersheim, L. Viganò, and D. Basin. Formalizing and analyzing sender invariance. In *Proceedings of FAST'06*, LNCS 4691, pages 80–95. Springer-Verlag, 2007.
43. 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.
44. ITU-T Recommendation H.530: Symmetric Security Procedures for H.510 (Mobility for H.323 Multimedia Systems and Services), 2002.
45. ITU-T Recommendation H.530, Corrigendum 1, 2003. Corrected version of [44].
46. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proceedings of LPAR'00*, LNCS 1955, pages 131–160. Springer-Verlag, 2000.
47. D. Johnson, C. Perkins, and J. Arkko. RFC3775 – Mobility Support in IPv6, June 2004.
48. G. Lowe. A hierarchy of authentication specifications. In *Proceedings of CSFW'97*, pages 31–43. IEEE Computer Society Press, 1997.
49. G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

50. U. M. Maurer and P. E. Schmid. A calculus for security bootstrapping in distributed systems. *Journal of Computer Security*, 4(1):55–80, 1996.
51. C. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
52. J. K. Millen and F. Muller. Cryptographic protocol generation from CAPSL. Technical Report SRI-CSL-01-07, SRI International, 2001.
53. J. K. Millen and V. Shmatikov. Constraint Solving for Bounded-Process Cryptographic Protocol Analysis. In *Proceedings of CCS'01*, pages 166–175. ACM Press, 2001.
54. S. Mödersheim. *Models and Methods for the Automated Analysis of Security Protocols*. PhD Thesis, ETH Zurich, 2007.
55. S. Mödersheim. On the Relationships between Models in Protocol Verification. *Information and Computation*, 206(2–4):291–311, 2008.
56. S. Mödersheim. Algebraic Properties in Alice and Bob Notation. In *Proceedings of Ares'09*, pages 433–440. IEEE Xplore, 2009. Extended version: Technical Report RZ3709, IBM Zurich Research Lab, 2008, `domino.research.ibm.com/library/cyberdig.nsf`.
57. S. Mödersheim and L. Viganò. Secure Pseudonymous Channels. In *Proceedings of Esorics'09*, to appear. Extended version: Technical Report RZ3724, IBM Zurich Research Lab, 2009, `domino.research.ibm.com/library/cyberdig.nsf`.
58. S. Mödersheim, L. Viganò, and D. Basin. Constraint Differentiation: Search-Space Reduction for the Constraint-Based Analysis of Security Protocols. *Journal of Computer Security*, to appear.
59. L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.
60. D. Peled. Ten Years of Partial Order Reduction. In *Proceedings of CAV 1998*, LNCS 1427, pages 17–28. Springer-Verlag, 1998.
61. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols*. Addison Wesley, 2000.
62. The Strand Space Method. `http://www.mitre.org/tech/strands/`.
63. M. Turuani. The CL-Atse Protocol Analyser. In *Proceedings of RTA'06*, LNCS 4098, pages 277–286. Springer-Verlag, 2006.