The Wolf Within *

Iliano Cervesato

Advanced Engineering and Sciences Division, ITT Industries, Inc.
Alexandria, VA 22303-1410 — USA
iliano@itd.nrl.navy.mil

Abstract

A formal specification of a security protocol cannot be limited to listing the messages exchanged. In *MSR*, each construct is associated with typing and data access specification (DAS) rules, which describe under which circumstances a principal can access keys and other information. A protocol specification is completed with a description of the intruder in the style of Dolev and Yao, the wolf in the protocol world. In this paper, we show that the protocol determines the intruder: the wolf is deep within. More precisely we show that the Dolev-Yao intruder rules can be automatically reconstructed from the DAS rules, and that the DAS rules can themselves be inferred from annotated typing declarations for the various message constructors.

1 Introduction

Most systems for the analysis of security protocols take as input an encoding of a protocol, an allocation of initial data, and a specification of the capabilities of the intruder in the style of Dolev-Yao [DY83, NS78]. A few rely on a hardwired attacker model. Even less do without any intruder model (the spi-calculus [AG99] is the only formalism with this property we are aware of). This would be a matter of taste if it were not for the fact that different protocols may require subtly different Dolev-Yao intruders for their analysis [Cer01a].

So far, the language *MSR* [Cer01c, Cer01d] belonged to the first class above. In this paper, we show that, in this language, the intruder can be automatically reconstructed from a (slightly annotated) specification of the protocol at hands: the wolf is within! We do so in two steps: first we show that the intruder rules can be recovered from the Data Access Specification (DAS) rules for a protocol, which precisely describe what information can be accessed, interpreted, and constructed by which principals. Then, we describe how simple annotations of the declarations of types and constructors are sufficient to reconstruct the DAS rules themselves.

We believe the main contribution of this work lies in making explicit the connection between the semantics of a specification language for cryptographic protocols and the expression of the Dolev-Yao intruder it supports. We conjecture that the ideas presented

* Partially supported by NRL under contract N00173-00-C-2086.

here apply not only to *MSR*, but to any specification language with an explicit intruder model, as long as its behavior is described in a sufficiently precise fashion (possibly, but not necessarily, with DAS rules). Languages that do not specify the intruder, such as the spi-calculus [AG99], have the cryptography and message manipulation completely defined in their semantics. This allows modeling the intruder as a generic process in parallel with the protocol (in the spi-calculus an attacker becomes just a tester on may-testing equivalence, with no restriction). Therefore, the present work suggests that the verification techniques available in the spi-calculus could be ported to *MSR*.

This paper is organized as follows: in Section 2 we briefly introduce *MSR*, the notion of DAS, and the *MSR* presentation of the Dolev-Yao intruder. In Section 3, we show how the intruder specification can be faithfully reconstructed from the DAS rules. In Section 4 we propose a technique for deriving the DAS rules from an annotated specification for a protocol. Section 5 hints at directions of future work.

2 Background

MSR originated as a simple logic-oriented language aimed at investigating the decidability of protocol analysis within the Dolev-Yao model [CDL$^+$99]. It evolved into a precise, powerful, flexible, and still relatively simple framework for the specification of complex cryptographic protocols, possibly structured as a collection of coordinated subprotocols [Cer01c, Cer01d]. It uses strongly-typed multiset rewriting rules over first-order atomic formulas to express protocol actions and relies on a form of existential quantification to symbolically model the generation of nonces and other fresh data. In particular, the user is given the option to use dependent types to express the association between keys and their owners: for example "$k : \mathsf{shK}\ A\ B$" declares k as a shared key between principals A and B. An earlier version of *MSR* fuels the *CAPSL* authentication protocol verification tool in the form of the underlying *CIL* intermediate language [DM99].

Strong typing opens the doors to a number of useful static checks. The most obvious, type checking, can be used to catch such specification errors as encrypting a message with a nonce or transmitting a long term key (assuming an appropriate set up of the type system). Another useful check is Data Access Specification (abbreviated DAS), which enforces such sensible requirements as, for example, that a principal may access the public key of any other principal, but in general not their private keys. DAS emerged as an attempt to formalize an often overlooked aspect of the Dolev-Yao model of security [DY83, NS78], on which *MSR* is based: the mechanisms by which auxiliary data are acquired, stored and maintained is kept abstract in this model, giving a principal what looks like magic access to the information he/she needs to execute the protocol. While this is inconsequential in the case of the name of other principals, care should be taken for more sensitive data such as keys: for example, A should have free access to her keys, but not to anybody else's. DAS specifies which information can be accessed by whom, and will reject any protocol specification that does not comply.

DAS is protocol-specific, although protocols relying on the same primitives and assumptions will generally share the same DAS. It is expressed through a number of judgments similar to typing judgments, each specified by means of inference rules, again similar to typing rules. For example, the following inference figure describes how to check a rewrite rule $r = lhs \rightarrow rhs$:

$$\frac{\Gamma; \cdot \Vdash_A lhs > \cdot \gg \Delta \quad \Gamma; \Delta \Vdash_A rhs}{\Gamma \Vdash_A lhs \rightarrow rhs} \textbf{das\_rule}$$

Here A is the principal executing this rule (the rule owner), the context Γ lists the type of every symbol appearing in r, and the knowledge set Δ accumulates all the information that A can deduce from the left-hand side lhs (incoming data) to construct the right-hand side rhs (outgoing data). A selection of other rules will be given in Section 3. When read algorithmically, a DAS verification consists of five parts:

1. Interpretation of the messages in the left-hand side of a rule (*e.g.* the cleartext of an encrypted component);
2. Access to data needed to interpret the left-hand side (typically keys);
3. Generation of data on the right-hand side (*e.g.* nonces);
4. Construction of messages on the right-hand side (*e.g.* for transmission over the network);
5. Access to data on the right-hand side (*e.g.* an interlocutor's name or public key).

In *MSR*, an intruder is just a distinguished principal, call it I, and its capabilities are any set of rewrite rules that only he/she can execute. The Dolev-Yao intruder [DY83, NS78] is a specific instance of this model which provides rules only for the most basic capabilities, *e.g.* intercepting a network message, opening an encrypted message with a known key, accessing information as mandated by DAS, etc. We have proved that an arbitrary attacker specification can be emulated by the Dolev-Yao intruder [Cer01b]. We have also shown that aspects of the Dolev-Yao intruder are protocol-dependent: for example, the ability to generate new keys only makes sense for protocols where a legitimate principal (*e.g.* the server of a key distribution protocol) generates keys.

3 DAS Rules $\Rightarrow$ Dolev-Yao Intruder Model

The capabilities of the Dolev-Yao intruder can be divided into five groups:

1. Interpretation of incoming messages;
2. Access to keys needed to interpret incoming messages;
3. Generation of new data;
4. Construction of outgoing messages;
5. Access to data needed to construct outgoing messages.

There is a striking correspondence with the five phases of DAS verification. We will show in this section that this is not a coincidence: the *MSR* rules specifying the Dolev-Yao intruder can be automatically reconstructed from the DAS rules that govern access for a given protocol. Due to lack of space, we will proceed by analyzing a number of significant examples, pretending to discover our transformation as we proceed. A formal account of this transformation is the subject of a forthcoming paper. We will go backwards, from item (5) to item (1) of the above two lists.

Access to a principal name in the right-hand side of a protocol rule is among the simplest inference figures of DAS: rule **das_axr_princ** below specifies that the rule owner A can access the name of any principal B. The Dolev-Yao intruder rule, **IPR**, that looks up a principal name is given next to it.

$$\frac{}{(\Gamma, B : \mathsf{principal}) \looparrowright_A B}\ \textbf{das\_axr\_princ}$$

$$\Longrightarrow \quad \textbf{IPR}\text{:}\ \big(\forall B : \mathsf{principal}.\quad \cdot \quad \rightarrow \quad \mathsf{I}(B)\big)^{\mathsf{I}}$$

There is a fairly straightforward transformation from **das_axr_princ** to **IPR**: the acting principal A is instantiated to I, selected meta-variables of **das_axr_princ** (B) are mapped to variables in **IPR**, the accessed data B is memorized in I's local store as $\mathsf{I}(B)$, and the context of the inference figure provides necessary typing information (B : principal) to the intruder rule (the rest is discarded). We check out this strategy on another right-hand side data access rule, one for shared keys: a principal A can look up a shared key k only if he/she is one of its co-owners (here if $k : \mathsf{shK}\ A\ B$).

$$\frac{}{(\Gamma, B : \mathsf{principal}, k : \mathsf{shK}\ A\ B) \looparrowright_A k}\ \textbf{das\_axr\_shk1}$$

$$\Longrightarrow \quad \textbf{IS1}\text{:}\ \begin{pmatrix} \forall B : \mathsf{principal}. & & & \\ \forall k : \mathsf{shK}\ \mathsf{I}\ B. & \cdot & \rightarrow & \mathsf{I}(k) \end{pmatrix}^{\mathsf{I}}$$

The corresponding intruder rule, displayed on the right, is reconstructed by applying the above four steps. Of course, we need to propagate the instantiation of A to I to all the occurrences of this variable. The dual rules, where $k : \mathsf{shK}\ B\ A$, are handled similarly. We gain confidence by observing that this technique transforms the rules for public and private asymmetric keys as expected:

$$\frac{}{(\Gamma, k' : \mathsf{privK}\ k, k : \mathsf{pubK}\ A) \looparrowright_A k'}\ \textbf{das\_axr\_privk}$$

$$\Longrightarrow \quad \textbf{IPV}\text{:}\ \begin{pmatrix} \forall k : \mathsf{pubK}\ \mathsf{I}. & & & \\ \forall k' : \mathsf{privK}\ k. & \cdot & \rightarrow & \mathsf{I}(k') \end{pmatrix}^{\mathsf{I}}$$

$$\frac{}{(\Gamma, B : \mathsf{principal}, k : \mathsf{pubK}\ B) \looparrowright_A k}\ \textbf{das\_axr\_pubk}$$

$$\Longrightarrow \quad \textbf{IPB}\text{:}\ \begin{pmatrix} \forall B : \mathsf{principal}. & & & \\ \forall k : \mathsf{pubK}\ B. & \cdot & \rightarrow & \mathsf{I}(k) \end{pmatrix}^{\mathsf{I}}$$

Working our way backwards from the consequent to the antecedent of an *MSR* rule, we will now look at the DAS (and intruder) rules that describe the construction of messages (item (4) of the above list). As a starter, we look at the rules that specify message concatenation:

$$\frac{\Gamma; \Delta \looparrowright_A t_1 \quad \Gamma; \Delta \looparrowright_A t_2}{\Gamma; \Delta \looparrowright_A t_1\, t_2}\,\mathbf{das\_cons\_conc}$$

$$\Longrightarrow \quad \mathbf{CMP}\text{:} \left(\forall t_1, t_2 : \mathsf{msg}. \quad \begin{matrix} \mathsf{I}(t_1) \\ \mathsf{I}(t_2) \end{matrix} \quad \rightarrow \quad \mathsf{I}(t_1\, t_2) \right)^{\mathsf{I}}$$

Aspects of the transformation devised so far apply here as well, namely the instantiation of the rule owner A to I and the mapping of meta-variables to rule variables, but new techniques are also required: the premises of rule $\mathbf{das\_cons\_conc}$ become the antecedent of the intruder rule, while the conclusion is associated to its consequent. The type of the (meta-)variables t_1 and t_2 is not specified in the context Γ: an auxiliary typing derivation for the term $(t_1\, t_2)$ relative to Γ is instead needed to deduce that their type must be msg. The knowledge set Δ is not used here: we will see that it can be thought of as a compendium of the intruder's knowledge. We test this extended transformation on the rules that handle the construction of messages encrypted with shared keys:

$$\frac{\Gamma; \Delta \looparrowright_A t \quad \Gamma; \Delta \looparrowright_A k}{\Gamma; \Delta \looparrowright_A \{t\}_k}\,\mathbf{das\_cons\_ske}$$

$$\Longrightarrow \quad \mathbf{SEC}\text{:} \left(\begin{matrix} \forall B, B' : \mathsf{principal}. \\ \forall k : \mathsf{shK}\, B\, B'. \\ \forall t : \mathsf{msg}. \end{matrix} \quad \begin{matrix} \mathsf{I}(t) \\ \mathsf{I}(k) \end{matrix} \quad \rightarrow \quad \mathsf{I}(\{t\}_k) \right)^{\mathsf{I}}$$

The same transformation applies to the other message construction figures for the right-hand side of a rule.

Next, we move to the generation of data (item (3) above). We have the following DAS and intruder rules:

$$\frac{(\Gamma, n : \mathsf{nonce}); (\Delta, n) \Vdash_A rhs}{\Gamma; \Delta \Vdash_A \exists n : \mathsf{nonce}.\, rhs}\,\mathbf{das\_gen\_nnc}$$

$$\Longrightarrow \quad \mathbf{GNC}\text{:} \left(\quad \cdot \quad \rightarrow \quad \exists n : \mathsf{nonce}. \quad \mathsf{I}(n) \right)^{\mathsf{I}}$$

Here, the generated object n augments the information stored in the knowledge set Δ. Such additions correspond to memorizing this object in the intruder's storage as $\mathsf{I}(n)$. Other data generation rules are handled similarly.

We reach the left-hand side of an *MSR* rule with item (2). We illustrate our transformation technique on one of the rules that accesses a shared key:

$$\frac{}{(\Gamma, B : \mathsf{principal}, k : \mathsf{shK}\, A\, B, \Gamma'); \Delta \Vdash^{s}_{A} k \gg (\Delta, k)}\,\mathbf{das\_axl\_shk1}$$

$$\Longrightarrow \quad \mathbf{IS1}\text{:} \left(\begin{matrix} \forall B : \mathsf{principal}. \\ \forall k : \mathsf{shK}\, \mathsf{I}\, B. \end{matrix} \quad \cdot \quad \rightarrow \quad \mathsf{I}(k) \right)^{\mathsf{I}}$$

Note that, again, the change to Δ expresses the acquisition of new information. This addition is realized at the intruder level by adding k in his private knowledge base as $\mathsf{I}(k)$. Observe that we already encountered the target rule $\mathbf{IS1}$ when devising a transformation for the DAS rules that govern the access to data on the right-hand side (item (5)). Other left-hand side data access rules are handled in a similar fashion, and have this property.

Next, we move to item (1) which handles the DAS rules that interpret information received in the left-hand side of an *MSR* rule. Let us look at how terms encrypted with a shared key are processed:

$$\frac{\Gamma; \Delta \Vdash^{s}_{A} k \gg \Delta' \quad \Gamma; \Delta' \Vdash_{A} t, \vec{t} \gg \Delta''}{\Gamma; \Delta \Vdash_{A} \{t\}_k, \vec{t} \gg \Delta''}\,\mathbf{das\_int\_ske}$$

$$\Longrightarrow \quad \mathbf{SDC}\text{:} \begin{pmatrix} \forall B, B' : \mathsf{principal}. & \mathsf{I}(\{t\}_k) & \rightarrow & \mathsf{I}(t) \\ \forall k : \mathsf{shK}\ B\ B'. & \mathsf{I}(k) & & \\ \forall t : \mathsf{msg}. & & & \end{pmatrix}^{\mathsf{I}}$$

The mapping of the objects appearing in the DAS rule on the left to the intruder rule on the right is explained by the techniques we have used so far. In particular, the choice of which terms should appear in the antecedent and in the consequent of the intruder rule are guided by the flow of knowledge acquisition: given $\{t\}_k$ with respect to Δ, we need to have access to k (possibly augmenting our knowledge to Δ') in order to deduce the term t. In general, the premise that produces the final knowledge set (Δ'' here) are kept as the intruder rule consequent.

A systematic application of the technique exemplified above to DAS rules of different classes yields the specification of the entire Dolev-Yao intruder, except for the rule that allows it to discard knowledge. This rule is however optional in the sense that any sequence of transitions that rely on it can be emulated without it, up to the addition of previously known information. It also yields a number of additional rules that are either redundant, *e.g.*

$$(\forall t : \mathsf{msg}.\ \mathsf{I}(t) \rightarrow \mathsf{I}(t))^{\mathsf{I}},$$

or innocuous but sensible, *e.g.*

$$(\forall t_1, \ldots, t_n : \mathsf{msg}.\ M'_{\mathsf{I}}(t_1, \ldots, t_n) \rightarrow \mathsf{I}(t_1), \ldots, \mathsf{I}(t_n))^{\mathsf{I}},$$

for some persistent memory predicate M'_{I} belonging to the intruder.

4 Typed Specification $\Rightarrow$ DAS Rules

In the previous section, we have informally outlined how the Dolev-Yao intruder can be automatically inferred from the DAS rules for a protocol. Since the DAS rules are to some extent protocol dependent (although often with many similarities), it is natural to ask whether these rules can themselves be produced automatically from an *MSR* specification of a protocol. Since they are syntax-directed, one may expect to be able to deduce them from the declarations for the types and constructs used to model a specific protocol. This is in general insufficient since constructs with identical declarations may have different DAS rules: for example public/private keys for encryption and verification/signature keys for keyed message digests (and consequently for digital signatures) are declared identically, and the declarations for the operations they support (public-key encryption and message digest) share the same structure. The next idea suggests annotating those declarations with sufficient information to reconstruct the rules. We will follow this path.

Consider first the DAS rules that describe data access on the right-hand side. We observe that they can be inferred by marking the declaration of the type of the different objects. Below, we prefix the declaration for principal with $\boxed{+}$ to indicate that there is free access to principal names:

$$\mathsf{principal} : \boxed{+}\,\mathsf{type}.$$

Here "type" indicates that "principal" is a type.

Some care should be taken in the case of dependent types since the accessibility of a piece of data may depend on properties the objects its type depends on. In the case of keys, it is crucial to be able to say that a given argument should be the rule owner. We indicate this with $\underline{\mathsf{principal}}$. We also write $\boxed{*}$ to signify that the accessibility of an argument is irrelevant. Then, we can annotate the declarations for private/public keys as follows:

$$\mathsf{pubK}\ \_ : \boxed{*}\,\mathsf{principal} \rightarrow \boxed{+}\,\mathsf{type}.$$
$$\mathsf{privK}\ \_ : \Pi A : \boxed{+}\,\underline{\mathsf{principal}}.\boxed{+}\,\mathsf{pubK}\ A \rightarrow \boxed{+}\,\mathsf{type}.$$

Here, "pubK " is declared as a dependent type, *i.e.* an object that is a type when applied to an argument of the proper type (a principal in this case). The declaration for "privK " is additionally parametric in the principal A mentioned in its argument. This is expressed with the type-theoretic construct Π, which can be thought of as a typed universal quantifier.

The DAS rules for term construction do not require any special provision since all their subterms must be constructible or accessible. Data generation is instead more tricky since some type of objects can be generated, while others cannot. We tag the former with $\boxed{!}$. Below are the declarations for nonces and shared keys. Notice how the duplication for the latter declaration.

$$\mathsf{nonce} : \boxed{!}\,\mathsf{type}.$$
$$\mathsf{shK}\ \_\ \_ : \boxed{+}\,\underline{\mathsf{principal}} \rightarrow \boxed{+}\,\mathsf{principal} \rightarrow \boxed{!}\,\mathsf{type}.$$
$$\mathsf{shK}\ \_\ \_ : \boxed{+}\,\mathsf{principal} \rightarrow \boxed{+}\,\underline{\mathsf{principal}} \rightarrow \boxed{!}\,\mathsf{type}.$$

Data access on the left-hand side of a DAS rule is governed by the same principles as on the right-hand side, but the format of rules is slightly different, mostly because of the need to remember and propagate information in a knowledge set. Therefore, the above declarations do not need to be modified to reconstruct these rules.

The interpretation of incoming messages is done by pattern matching. Therefore, we simply need to mark which arguments of a term constructor can be deduced in this way (tag $\boxplus$ — overloaded), which ones should be provided (tag $\boxminus$), and which ones do not need to be provided but are not deduced either (tag $\boxed{/}$). We first show declarations for the constructors described or implied in the above presentation, namely concatenation ($\_\,\_$), shared-key encryption ($\{\_\}_\_$), and public-key encryption ($\{\!\{\_\}\!\}_\_$):

$$\begin{aligned}
&_\,_ : \boxminus\,\mathsf{msg} \to \boxminus\,\mathsf{msg} \to \mathsf{msg}.\\
&\{_\}__ : \boxminus\,\mathsf{msg} \to \Pi A : \boxed{/}\,\mathsf{principal}.\Pi B : \boxed{/}\,\mathsf{principal}.\boxplus\,\mathsf{shK}\ A\ B \to \mathsf{msg}.\\
&\{\!\{_\}\!\}__ : \boxminus\,\mathsf{msg} \to \Pi A : \boxed{/}\,\mathsf{principal}.\Pi k : \boxed{/}\,\mathsf{pubK}\ A.\boxplus\,\mathsf{privK}\ k \to \mathsf{msg}.
\end{aligned}$$

The first declaration simply affirms that, upon receiving a concatenated message $(t_1\, t_2)$, a principal can always inspect the components t_1 and t_2. The second declaration, which describes shared-key encryption, is more interesting: given a message of the form $\{t\}_k$, it allows accessing the subterm t (tagged $\boxminus$) only if the argument k is known (as indicated by tagging $\mathsf{shK}\ A\ B$ with $\boxplus$). Observe that the tag $\boxed{/}$ implies that nothing is revealed about A and B, and that access to these terms is not necessary. The third rule, intended to describe how public-key decryption is handled, operates in a similar way: observe that the principal performing the decryption is required to know the private key ($\boxplus$), but not necessarily its public cognate ($\boxed{/}$).

It is interesting to describe how our tagging scheme performs on other standard cryptographic constructions. We will now focus on hashing ($\mathsf{hash}\ t$) and below on digital signatures. Other constructs could be handled similarly. A hash function is declared as follows:

$$\mathsf{hash}\ \_ : \boxplus\,\mathsf{msg} \to \mathsf{msg}.$$

This declaration requires the argument of hash to be known (it is tagged with $\boxplus$). Therefore, pattern matching can be used at most to test that a hashed object is equal to some previously known value.

For compactness, digital signatures are often modeled as expression of the form "$[t]_k$", which stands for both the transmitted term t and the keyed digest of t with signature key k. It is convenient to take a more explicit approach: we write $\mathsf{md}_k(t)$ for the message digest of t with respect to k. Therefore, a message t together with its digest $\mathsf{md}_k(t)$ is rendered as the concatenation $[t]_k = (t\ \ \mathsf{md}_k(t))$. We then have the following dual declaration for message digests:

$$\mathsf{md}\_(\_) : \boxplus\,\mathsf{msg} \to \Pi A : \boxed{/}\,\mathsf{principal}.\Pi k : \boxed{/}\,\mathsf{sigK}\ A.\boxplus\,\mathsf{verK}\ k \to \mathsf{msg}.$$

This is reminiscent of hash functions: given the verification key, it is possible at most to test the equality of the signed message and of a previously known term. Whenever a principal wants to access a signed message without verifying its digest, he/she can assign the generic type msg to the later.

Observe that reconstructing DAS rules for data access and generation required annotating the classification of *MSR* types (*e.g.* principal, shK and nonce). The other DAS rules are instead derived from annotations on the classification of *MSR* term constructors (the operations of encryption, concatenation, etc.).

The above annotations implement a trimmed down form of the DAS rules, but with sufficient detail to allow reconstructing them. In this sense, they add semantic information to otherwise purely syntactic declarations. Again, a formal description of this process is the subject of a forthcoming paper.

The next natural question is whether it is possible to reconstruct these annotations from the untagged declarations and a protocol specification. The answer is affirmative, although in general several sets of annotations may be revealed (and the user will have to choose the one that corresponds to his/her intentions). In order to do so, observe that there is only a finite number of possible annotations of an untagged set of declarations. It then suffices to validate each of them against the protocol specification and keep the most restrictive ones. Of course, this way of proceeding has an exponential cost. It is plausible that there are more efficient tagging algorithms, but this has not been investigated yet.

5 Final Remarks

We have described how the *MSR* specification of the Dolev-Yao intruder can be extracted from the DAS rules for a protocol specification, which govern data accessibility, and how these rules can themselves be deduced from annotated declarations for the constructs that appear in the protocol.

Although an interesting result by itself, it raises a number of questions. First of all, we believe that our reliance on *MSR* is not crucial in this results: a Dolev-Yao intruder should be derivable in any sufficiently expressive protocol specification language (possibly after defining DAS-like rules as an intermediary step). This has however to be shown in practice. Second, a few languages such as the spi-calculus [AG99] do not specify an intruder. It would be very interesting to formalize their relationship to *MSR* in the light of the results in this paper, and to adapt the proof techniques of the spi-calculus to *MSR*. Finally, Dolev-Yao like intruder specifications are unoptimized and, due to their atomicity, can cause unnecessary overhead when reasoning about the security goals of a protocol. We would like to be able to automatically construct optimized intruder rules, possibly from a DAS derivations for the protocol specification at hands.

Acknowledgments

We are grateful to the anonymous reviewers whose comments contributed to producing a substantially better final version of the manuscript,

References

[AG99] Martí Abadi and Andrew Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[CDL+99] Iliano Cervesato, Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In P. Syverson, editor, *Proc. CSFW'99*, pages 55–69, Mordano, Italy, 1999. IEEE Computer Society Press.

[Cer01a] Iliano Cervesato. Data access specification and the most powerful symbolic attacker in *MSR*. Extended version of [Cer01b]. Unpublished manuscript, 2001.

[Cer01b] Iliano Cervesato. The Dolev-Yao intruder is the most powerful attacker. In J. Halpern, editor, *LICS'01*, Boston, MA, 2001. IEEE Computer Society Press. Short paper.

[Cer01c] Iliano Cervesato. Typed MSR: Syntax and examples. In V.I. Gorodetski, V.A. Skormin, and L.J. Popyack, editors, *Proc. MMM'01*, pages 159–177, St. Petersburg, Russia, 2001. Springer-Verlag LNCS 2052.

[Cer01d] Iliano Cervesato. Typed multiset rewriting specifications of security protocols. In A. Seda, editor, *Proc. MFCSIT'00*, Cork, Ireland, 19–21 July 2001. Elsevier ENTCS.

[DM99] Grit Denker and Jonathan K. Millen. CAPSL Intermediate Language. In N. Heintze and E. Clarke, editors, *Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP*, Trento, Italy, July 1999.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, 1983.

[NS78] Roger M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.