Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science 128 (2005) 55–63
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Tags for Multi-Protocol Authentication

Matteo Maffei [1],[2]

Dipartimento di Informatica
Università Ca'Foscari
Venezia, Italia

Abstract

Formal methods have been proved successful in analyzing different kinds of security protocols. They typically formalize and study the security guarantees provided by cryptographic protocols, when executed by a (possibly unbounded) number of different participants. A key problem in applying formal methods to cryptographic protocols, is the study of *multi-protocol systems*, where different protocols are concurrently executed. This scenario is particularly interesting in a global computing setting, where several different security services coexist and are possibly combined together. In this paper, we discuss how the tagging mechanism presented in [4,5] addresses this issue.

Keywords: Security, Process Calculi, Static Analysis

1 Introduction

Security protocols are designed to provide diverse security guarantees in possibly hostile environments: typical guarantees include the secrecy of a message exchange between two trusted entities, the freshness and authenticity of a message, the authenticity of a claimed identity, ... and more. The presence of hostile entities makes protocol design complex and often error prone, as shown by many attacks to long standing protocols reported in the literature (see, e.g., [6,7,13,18,19]). In most cases, such attacks dwell on flaws in the protocols' logic, rather than on breaches in the underlying cryptosystem. Indeed, even when cryptography is assumed as a fully reliable building-block,

[1] Work partially supported by EU Contract IST-2001-32617 'Models and Types for Security in Mobile Distributed Systems' (MyThS).
[2] Email: maffei@dsi.unive.it

doi:10.1016/j.entcs.2004.11.042

an intruder can engage a number of potentially dangerous actions, notably, intercepting/replaying/forging messages, to break the intended protocol invariants. Formal methods have proved very successful as tools for protocol design and validations.

On the one hand, several model-checking techniques have been applied to analyze security protocols (see, e.g., [17,19]), leading to the discovery of several attacks. However, their precision is balanced by the necessity to explore all the possible message sequences, generated during the execution of the protocol. Therefore the analysis of an unbounded number of participants, possibly running different and unknown protocols, turns out to be infeasible. On the other hand, static analysis techniques like, e.g., type systems [1,9,10] and control flow [2,3], aim at proving a property of a program (a protocol in our case) by only inspecting the source code. They are appealing because (i) only the code is inspected and there is no need of generating and exploring all the possible execution sequences; (ii) by exploiting compositionality, it becomes feasible to (even automatically) prove correctness of protocols with unbounded number of sessions and participants. Compositionality is also appealing for studying multi-protocol systems, since it might give local guarantees of correctness, without analyzing the system as a whole. These advantages are payed by approximating the property of interest thus loosing precision: static analyses are typically sound (an incorrect protocol is never validated) but not complete, as there might be false-negative, i.e., correct protocols that cannot be validated.

Typically, the approaches mentioned above study the interactions among principals running the *same* protocol in an untrusted environment. The interaction among *different* protocols, possibly carrying out some common subtasks, is considered an open issue [16], and it is particularly interesting in a global computing setting, where several security services coexist and are possibly combined together. As a matter of fact, this kind of interaction may be tricky. As an example, let us consider the two following simple protocols:

$$\begin{array}{ll} \alpha.1)\ B \rightarrow A : n & \beta.1)\ A \rightarrow B : A, M \\ \alpha.2)\ A \rightarrow B : \{B, M, n\}_{k_{AB}} & \beta.2)\ B \rightarrow A : \{B, n, M\}_{k_{AB}} \\ & \beta.3)\ A \rightarrow B : n \end{array}$$

The first protocol is the well-known ISO-Two-Pass Authentication protocol: B sends out a nonce, namely a random challenge, which is encrypted by A together with the identity label of B and the message she is willing to authenticate. The encryption key k_{AB} is a key shared between A and B. When

B receives that message, and checks its freshness by verifying the equality of the nonces, he is guaranteed that M is authentic from A. In protocol β, A sends as cleartext a message M, for instance a broadcast message, and her identity label. In the second message, B asks A to authenticate that message, by sending a ciphertext, encrypted with a long term key shared with A, containing the identity label B , a fresh nonce n and the message M. A may confirm by publishing n.

The two protocols are safe when they use different keys. However, they are affected by the following attack when the same key is exploited for carrying on both the protocols:

$$\begin{aligned}
&\alpha.1)\ B \rightarrow I(A) : n\\
&\beta.1)\ I(A) \rightarrow B : A, n\\
&\beta.2)\ B \rightarrow I(A) : \{B, n_A, n\}_{k_{AB}}\\
&\alpha.2)\ I(A) \rightarrow B : \{B, n_A, n\}_{k_{AB}}
\end{aligned}$$

B starts protocol α sending a fresh nonce for authenticating A, the enemy intercepts and reflects it, together with the identity label A, to another instance of B running protocol β . B mistakes the nonce for the broadcast message of protocol β and asks A to authenticate that message. The enemy reflects the third message to B who mistakes that message for the ciphertext of the second protocol, so authenticating A. Unfortunately, A is not present in the authentication task! The attack exploits the structural equality between the two ciphertexts.

In [4,5], we propose a static analysis for authentication, based on the ρ-spi calculus. The analysis is compositional: each principal is separately type-checked. The main theorem ensures that the parallel composition of correct principals is safe. Our notion of safety formalizes the agreement property proposed by Lowe [14]: "whenever *Bob* completes the initiator role, apparently with *Alice*, then *Alice* has previously been running the responder role, apparently with *Bob*, in the same authentication task. Moreover *Alice* and *Bob* agree on the message m to be authenticated".

Compositionality is achieved by exploiting a tagging mechanism. This suffices for expressing the authentication guarantees carried out by each ciphertext, regardless to the particular authentication protocol originating it. In this paper we discuss how that tagging mechanism applies to multi-protocol systems.

2 Tags for Authentication

Table 1 Tagging Scheme

	Protocol	I	R	k
POSH	$B \rightarrow A : n$ $A \rightarrow B : \{\mathsf{Id}(I), \mathsf{Auth}(M), R(n)\}_k$	A	Claim	$key(A,B)$
		B	Verif	$key(A,B)$
		B	Verif	$key_{priv}(A)$

	Protocol	I	R	k
SOPH	$B \rightarrow A : \{\mathsf{Id}(I), \mathsf{Auth}?(M), R_{public}?(n)\}_k$ $A \rightarrow B : n$	A	Claim	$key(A,B)$
		B	Verif	$key(A,B)$
		B	Verif	$key_{pub}(A)$

	Protocol	I	R	$k_I(I = A, B)$
SOSH	$B \rightarrow A : \{\mathsf{Id}(I), \mathsf{Auth}?(M_B), R_{secret}?(n)\}_{k_A}$ $A \rightarrow B : \{R_{secret}(n), \mathsf{Auth}(M_A)\}_{k_B}$	A	Claim	$key(A,B)$
		B	Verif	$key(A,B)$
		B	Verif	$key_{pub}(I)$

Messages used in authentication protocols are often ambiguous. For instance, let us consider the following simple ciphertext: $\{A, n, m\}_{k_{AB}}$. A first ambiguity regards the role of each message component: n might be a nonce and m a message to be authenticated, but also the opposite might be true. Even when a single protocol is executed, this kind of ambiguity often causes attacks, referred to as *type-flaws*. Furthermore, a second, more tricky, level of ambiguity is possible: the ciphertext might be sent from A to B for authenticating M (as in protocol α) or for asking B whether or not he agrees on M (for instance, M might be a fresh session key created by A); but, it might be sent also from B to A with the same goals. This ambiguity often causes attacks when multiple sessions of a single protocol are executed but, of course, the situation is even worse in multi-protocol systems: as discussed by Syverson and Meadows in [15], "problems can arise when a protocol is interacting with another protocol that does not use a tagging scheme, or tags data in a different way". We propose a tagging mechanism, uniformly applicable to different authentication protocols, preventing both the forms of ambiguities.

Ciphertexts may contain an identity label, which can specify the *claimant* or the *verifier* of the authentication task. A nonce is typically used for guaranteeing the freshness of ciphertexts. Ciphertexts may also contain a message to be authenticated. Protocols are usually divided into three classes, according to the nonce-handshake [10,12]: in *Public-Out Secret-Home* ($POSH$), the nonce is sent as cleartext and received into a ciphertext, in *Secret-Out Public-Home* ($SOPH$) the opposite happens, while in *Secret-Out Secret-Home* ($SOSH$), the nonce is sent out and received back into (different) ciphertexts. Protocol α, in §1, uses a $POSH$ nonce handshake, while protocol β is based on a $SOPH$. As an example of $SOSH$ nonce handshake, let us consider the following protocol:

$$\gamma.1)\ B \rightarrow A : \{n, B, M_B\}_{k_A^+}$$
$$\gamma.2)\ A \rightarrow B : \{n, M_A\}_{k_B^+}$$

The first message is encrypted with the public key of A. By that ciphertext, B asks A whether or not she agrees on M_B and she is willing to authenticate herself with B, which is the verifier of the authentication session. This part of the nonce-handshakes is exactly the same as in $SOPH$ nonce handshakes. The second message differs, however, since the nonce is kept secret (it is encrypted with the public key of B and cannot be read by the enemy). By sending back that nonce, A confirms the request received from B, as in the $SOPH$ nonce-handshake, but also authenticates M_A which is originated by A and is sent encrypted together with the secret nonce. In this sense, the second message combines the properties of the $POSH$ and the $SOPH$ nonce-handshakes. Table 1 summarizes the three nonce-handshakes and how they can be typed according as the nonce-handshake.

POSH The ciphertext may be encrypted either by a key shared between A and B, or by the private key of A. In the former case, an identity label must appear for disambiguating the originator and the intended receiver of the ciphertext [3]. If the identity label is A, then the nonce is tagged by Claim, since A is the claimant of the authentication session. If it is B, then the nonce is tagged by Verif, since B is the verifier. In the latter case,

[3] The identity label and the nonce are necessary in order to prevent reflection and replay attacks, respectively (for details see [6,8]).

an identity label is required for specifying the intended verifier of the authentication session. Hence, the nonce is tagged by Verif.

SOPH The ciphertext may be encrypted either by a key shared between A and B, or by the public key of A[4]. Message components are tagged similarly to the previous case; tags are interrogative, since B is *asking* A whether or not she agrees on M and is willing to accept B as verifier of the authentication task. The subscript *public* specifies that the nonce is supposed to be sent as cleartext in case of confirm.

SOSH This nonce-handshake combines the two previous ones. The nonce is kept secret, so only A can send back that nonce: this motivates the subscript $secret$. When encrypting that nonce inside the second ciphertext, A may also authenticate a fresh message, which is tagged by Auth.

These patterns may be combined together since a ciphertext might contain, for instance, two nonces used in different kind of nonce-handshakes. Moreover, as shown in [4] for the $POSH$ nonce-handshake, this tagging scheme scales up when Trusted Third Parties (TTP) are involved in the protocol and when session keys are used for achieving authentication.

3 Tagging and Multi-Protocol Systems

The tagging mechanism proposed here solves the two kinds of ambiguities discussed in §2: nonces, messages and identity labels are univocally determined. Moreover, tags show who is the originator and the intended receiver of the ciphertext and whether it represents a challenge or an answer in the authentication task.

Moreover, tags explicitly indicate the authentication guarantees conveyed by ciphertexts. Indeed, they do not depend on the protocol originating them: assuming all the participants to agree on this tagging mechanism, tags allow a *local* reasoning. For instance, if A receives a ciphertext which is an answer to a challenge sent previously to B, then A may authenticate B, independently of the particular protocol B is running. The only assumption on that protocol is that it respects our tagging mechanism.

This form of compositionality, formalized in [4,5], allows us to reason on multi-protocol systems, where principals do not know anything on the world around them.

As an example, let us go back to the two protocols in §1: we argue that

[4] The private key of A cannot be used since the nonce is supposed to be sent back only in case of confirm by A: a ciphertext encrypted with the private key of A would be read by the enemy which might send back the nonce.

the ciphertext in protocol α can be tagged as $\{\mathsf{Id}(B), \mathsf{Auth}(M), \mathsf{Verif}(n)\}_{k_{AB}}$, since A is *specifying* B as verifier of the authentication task. In protocol β, the ciphertext can be tagged as $\{\mathsf{Id}(B), \mathsf{Verif}_{public}?(n),\ \mathsf{Auth}?(M)\}_{k_{AB}}$, since B is *asking* A whether she is willing to authenticate M with B, namely, the verifier of the authentication task. Notice that the type-flaw attack is not possible on the tagged version of the protocols, since message M and nonce n are tagged differently and the tag of the nonce shows that, in protocol α, the ciphertext represents a response, while in protocol β it represents a challenge.

An interesting point is that there might be different ways of tagging a protocol. For instance, let us consider a simple variant of protocol β in §1.

$$\beta'.1)\ B \rightarrow A : n$$
$$\beta'.2)\ A \rightarrow B : \{A, B, M, n\}_{k_{AB}}$$

The ciphertext can be tagged either as $\{\mathsf{Id}(A), B, \mathsf{Auth}(M), \mathsf{Claim}(n)\}_{k_{AB}}$ or as $\{A, \mathsf{Id}(B), \mathsf{Auth}(M),\ \mathsf{Verif}(n)\}_{k_{AB}}$. Notice that the tagging is necessary for preserving the safety in a multi-protocol system, where protocol β' is executed concurrently with another variant:

$$\beta''.1)\ B \rightarrow A : n$$
$$\beta''.2)\ A \rightarrow B : \{B, A, M, n\}_{k_{AB}}$$

Indeed, the untagged version of the two protocols is affected by the following attack:

$$\beta''.1)\ B \rightarrow I(A) : n$$
$$\beta'.1)\ \ I(A) \rightarrow B : n$$
$$\beta'.2)\ \ B \rightarrow I(A) : \{B, A, M, n\}_{k_{AB}}$$
$$\beta''.2)\ I(A) \rightarrow B : \{B, A, M, n\}_{k_{AB}}$$

4 Conclusion and Related Work

Multi-protocol systems are an emerging issue in the research on security protocols analysis [16]. Untrusted environments do not contain only Dolev-Yao intruders, but also trusted principals running different protocols. The interaction between different protocol sessions may be tricky, and difficult to analyze.

In [11], Guttman and Thayer prove, in the Strand Spaces formalism [12], that if the sets of encrypted messages that the different protocols handle are disjoint, then the concurrent execution of those protocols is safe. In other words, a sufficient condition for guaranteeing safety is the following: if a protocol uses a message, then no other protocol should construct a message of that form. Intuitively, in a global computing setting, it is necessary to avoid confusion among different protocols in order to safely implement security services.

Indeed, this condition can be simply implemented: protocol designers might agree to allow an independent body to choose their protocol-specific identifiers for them. Alternatively, designers might choose them randomly, following some common algorithm guaranteeing a negligible probability of collisions.

We approach the problem from the opposite perspective: we do not prevent interaction among different protocols. On the contrary, we prove [4,5] that the communication among different protocols agreeing on our tagging scheme results to be safe.

Tagged messages explicitly indicate the authentication guarantees they provide, so allowing a principal to accept authentication regardless of the protocol the other party is running. Moreover, also the generation of tagged messages is ruled by a local reasoning (see [4,5] for details). This may be thought of as a first step towards an ideal notion of protocol-independent communication, where the security goal is achieved by simply adhering to a shared tagged scheme.

Acknowledgements Thanks to the anonymous referees for their very helpful comments and suggestions and to Riccardo Focardi for invaluable feedback.

References

[1] M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. *Theor. Comput. Sci.*, 298(3):387–415, 2003.

[2] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, pages 126–140. IEEE Computer Society Press, June 2003.

[3] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis can find new flaws too. In *Proceedings of the Workshop on Issues on the Theory of Security (WITS'04)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2004.

[4] M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of authentication protocols. In *Proceedings of European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 140–154. Springer-Verlag, 2004.

[5] M. Bugliesi, R.Focardi, and M.Maffei. A theory of types and effects for authentication. In *ACM Proceedings of Formal Methods for Security Engineering: from Theory to Practice (FMSE 2004)*, pages 1–12. ACM Press, October 2004.

[6] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. http://www.cs.york.ac.uk/∼jac/papers/drareview.ps.gz, November 1997.

[7] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proceedings of ICALP'00*, pages 354–372. Springer LNCS 1853, July 2000.

[8] R. Focardi and M. Maffei. ρ-spi calculus at work: Authentication case studies. In *Proceedings of Mefisto Project*, volume 99 of *Electronic Notes in Theoretical Computer Science*, pages 268–293. Elsevier, August 2004.

[9] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 145–159. IEEE Computer Society Press, June 2001.

[10] A. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. In *Proceedings of 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 77–91. IEEE Computer Society Press, 24-26 June 2002.

[11] Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 24–34. IEEE Computer Society Press, July 2000.

[12] Joshua D. Guttman and F. Javier Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[13] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 255–268. IEEE Computer Society Press, July 2000.

[14] G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop (CSFW'97)*, pages 31–44. IEEE Computer Society Press, 1997.

[15] C. Meadows and P. Syverson. Formal specification and analysis of the group domain of intrepretation protocol using npatrl and the nrl protocol analyzer, 2003. to appear in Journal of Computer Security.

[16] Catherine Meadows. Open issues in formal methods for cryptographic protocol analysis. *Lecture Notes in Computer Science*, 2052:21–36, 2001.

[17] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.

[18] R M Needham and M D Schroeder. Authentication revisited. *ACM SIGOPS Operating Systems Review*, 21(1):7–7, 1987.

[19] L. C. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.