Principles for Entity Authentication*

Michele Bugliesi, Riccardo Focardi, and Matteo Maffei

Dipartimento di Informatica, Università Ca' Foscari di Venezia,
{michele,focardi,maffei}@dsi.unive.it

Abstract. We study the roles of message components in authentication protocols. In particular, we investigate how a certain component contributes to the task of achieving entity authentication. To this aim, we isolate a core set of roles that enables us to extract general principles that should be followed to avoid attacks. We then formalize these principles in terms of rules for protocol parties and we prove that protocols designed according to these rules will achieve entity authentication.

1 Introduction

Security protocols, also known as cryptographic protocols, are designed to reach specific security goals in (possibly) hostile environments, like, e.g., Internet. Typical goals include the communication of a secret between two trusted entities, an authenticated message exchange, the generation and the sharing of a session key, the authentication of an entity with respect to another entity (e.g., to a server), and more. The design of security protocols is complex and often error prone, as witnessed by the many attacks to long standing protocols reported in the recent literature on the subject (see, e.g., [6,8,13, 16,19,20]). Most of these attacks do not need to break cryptography to be performed. Indeed, even when cryptography is assumed as a fully reliable building-block, an intruder can engage a number of potentially dangerous actions: it can intercept messages before they reach their destination, insert new messages, read those that travel along the communication links and forge new ones using the knowledge it has previously gained. All these actions are available to an intruder willing to try and break a protocol by exploiting a flaw in the underlying protocol logic.

In this paper, we focus on (shared-key based) cryptographic protocols for *entity authentication*, i.e. on protocols that enable one entity to prove its claimed identity to another entity [14,18,9], and discuss a novel method to protect such protocols from attacks to their logic.

A typical source of flaws in security protocols, and specifically in authentication protocols, is a poor interpretation of messages, whereby certain messages are believed to convey more guarantees than they really do. As observed in [2] "every message should say what it means, i.e., its interpretation should depend only on its content". As a trivial example of a message exchange that fails to comply with this principle consider the protocol:

$$A \rightarrow B : \{\text{“Here I am!”}\}_{K_{AB}}$$

* Work partially supported by MIUR project "Modelli formali per la sicurezza" and EU Contract IST-2001-32617 "Models and Types for Security in Mobile Distributed Systems" (MyThS).

M. Broy and A.V. Zamulin (Eds.): PSI 2003, LNCS 2890, pp. 294–306, 2003.

in which *Alice* is sending to *Bob* the message "Here I am!" encrypted with a long-term key shared between them. When Bob receives the message, he could erroneously deduce that it has been generated by Alice since it is encrypted with a key that only Alice (besides Bob himself) knows. However, this is not true if Bob previously ran the same protocol with Alice (with exchanged roles). In that case, the following, well-known, *reflection attack* could be exploited by an intruder E:

$$\begin{array}{ll} a) & B \rightarrow E(A) : \{\text{“Here I am!”}\}_{K_{AB}} \\ b) & E(A) \rightarrow B : \{\text{“Here I am!”}\}_{K_{AB}} \end{array}$$

In the first run a, E pretends to be A (denoted with $E(A)$) and intercepts the message sent by Bob; in the second run b, $E(A)$ replays the same message back to Bob. As a consequence, B erroneously interprets his own message as sent by Alice. The problem is that Bob is assuming that the message has been generated by Alice without this being explicitly indicated in the message. A simple solution is to provide this additional information within the message, as in $A \rightarrow B : \{A, \text{“Here I am!”}\}_{K_{AB}}$, where the first component now signals that A is the 'claimant' of the protocol.

Motivated by this need to make the interpretation of messages unambiguous, we investigate the roles of message components in authentication protocols. In particular, we study how entity identifiers, associated with the entity roles in the protocols, contribute to the task of achieving entity authentication. We identify three fundamental roles: (i) *Claimant*, the role of a principal willing to authenticate itself to some other principal; (ii) *Intended verifier*: the role of a principal that a claimant wants to act as a verifier for its response; (iii) *Key owner*: the role of a principal that owns a session key. These roles apply well, and uniformly, for a wide class of entity authentication protocols.

We propose a number of principles to be followed by the trusted parties of a protocol. These principles formalize provably safe ways for generating new encrypted blocks containing roles – called *authentication blocks* – from existing ones, and provide a formal basis for reasoning on authentication in terms of the authentication guarantees that are provided by decrypting such special blocks. For example, a trusted server S can assign to an entity A the claimant role in a message directed to another entity B, provided it has enough information to guarantee to B that A is indeed the claimant, i.e., provided that S has decrypted an authentication block stating that A is willing to authenticate with B. Dually, if an entity B decrypts a block where A is playing the claimant role, then he is guaranteed that A is indeed the identity of the claimant and that A was recently running the authentication protocol with him, i.e., A is authenticated to B.

We formalize these principles in terms of rules for protocol parties and we prove that protocols designed according to these rules will achieve entity authentication.

The rest of the presentation is organized as follows. § 2 describes the general assumptions of the model we work with, and introduces the notion of authentication block. § 3 defines the design principles, formalized as inference rules, and illustrates them with several examples. § 4 concludes with final remarks and a discussion on related work.

2 Protocol Roles and Authentication Blocks

As we stated in the introduction, our focus is on protecting protocols from attacks that are based on flaws in the protocols' logic and do that not exploit any weakness of the underlying cryptographic primitives. We thus make the so called *perfect cryptography* assumption, i.e. we abstract away from security issues regarding cryptographic primitives and specific cryptographic algorithms, and we assume every cryptographic operation to be a secure building-block. Moreover we assume that keys are carefully managed, i.e., long-term keys are safely distributed to participants and can never be learned by any intruder and, similarly, session keys are never circulated within messages once they have been first distributed.

Authentication protocols can be partitioned into two classes depending on whether or not they are based on a *Trusted Third Party* (TTP), i.e., a trusted entity that acts as intermediate party for the authentication task. As we shall see, our principles allow us to deal with both kinds of authentication in a uniform way.

The following conventions are assumed throughout. The capital letters A and B denote protocol *principals*, which are distinguished from TTPs denoted by S. Instead, we use I and J to refer to generic participants in the protocol, either principals or TTP's. We use K and N to denote keys and nonces, respectively. The notation K_{IJ} indicates a long-term key shared between I and J. We assume that $K_{IJ} = K_{JI}$. Finally, V is used to denote either a key or a nonce and R ranges over the roles Claim, Verif and Owner.

As discussed earlier, we consider three possible roles that entities can play during the authentication task: *Claimant*, *Intended verifier*, and *Key owner*. There are other message components that are necessary to make these roles useful for the purpose of authentication. In particular, each role needs a time-dependent element, like e.g., nonces, timestamps, sequence numbers, that provides *freshness*. Here we only consider *nonces*, i.e., numbers that are used only once: they are typically generated as challenges, and sent back inside encrypted messages, to be *checked* and thus prove that a certain message is not a replay of an old one. Together with nonces, we associate entities with the session key they possess (this is always true of entities playing the *Key owner* role). This is useful for distribution of session keys, where a trusted server wants to communicate that a certain entity owns a certain session key.

Authentication blocks are abstractions of encrypted messages which only contain the message components that are directly relevant to authentication, plus the role of such components in the authentication task. Formally,

Definition 1. *An authentication block is a tuple of the form* $\langle \mathsf{R}, I, N, K \rangle_{K_{I_1 I_2}}$. *The session key* K *is mandatory only for the role* Owner*: when it is missing the block is denoted by* $\langle \mathsf{R}, I, N \rangle_{K_{I_1 I_2}}$.

The use of roles and nonces within authentication blocks conveys useful information and guarantees to any entity which successfully decrypts them. Specifically, assume that entity I_1 decrypts the authentication block $\langle \mathsf{R}, I, N, K \rangle_{K_{I_1 I_2}}$ and checks the nonce N (it had previously generated), and let $I_1 \neq I_2$. Then the following can be inferred: "Entity I is the owner of the fresh key K (if K is present), and is recently playing the role R with respect to I_1 and I_2", that is:

- if $\mathsf{R} = \mathsf{Claim}$, I_1 is not a TTP and $I_1 \neq I$ then I_1 is guaranteed that I has recently initiated an authentication session with I_1;
- if $\mathsf{R} = \mathsf{Verif}$, I_1 is a TTP and $I_1, I_2 \neq I$ then I_1 is guaranteed that I_2 has recently initiated an authentication session with I;
- if $\mathsf{R} = \mathsf{Owner}$ then I_1 is guaranteed that I knows the fresh session key K;

In all the remaining cases, no guarantee can be made based on the decryption of the block.

As we mentioned earlier, "checking the nonce" provides the freshness guarantees on authentication blocks that are needed to counter replay attacks. Also note that guarantees are provided by decrypting authentication blocks only in some specific situations. In particular, since the Claim role is used to authenticate I to I_1 (possibly through a TTP), we require I_1 to be a party other than a TTP and that $I_1 \neq I$, i.e., that I_1 does not accept authentication requests from itself (which is a typical source of "reflection attacks"). Since the Verif role is used by I_2 to communicate to a TTP I_1 that I_2 itself intends to authenticate with I, we ask that I_1 be a TTP and that I be different from the other two identities. Finally, the assumption that $I_1 \neq I_2$ implies that we disregard all issues related to self-authentication.

Example 1. We illustrate the use of authentication blocks in reasoning on the interpretation of messages and in making their interpretation unambiguous. Consider a message $\{M, A, N\}_{K_{BS}}$ encrypted with a long-term key shared between principal B and TTP S, and containing a message M, the entity identifier A and a nonce N. One can deduce that this message is probably used to authenticate something, as a nonce is present, but nothing more can be said from the message itself. If instead, we represent it as an authentication block, we recover a precise, and unambiguous semantics. For example we can represent $\{M, A, N\}_{K_{BS}}$ as the block $\langle \mathsf{Claim}, A, N \rangle_{K_{BS}}$, meaning that this message can be used to authenticate entity A to B, once nonce N has been checked by B. Alternatively, $\{M, A, N\}_{K_{BS}}$ could be mapped to $\langle \mathsf{Verif}, A, N \rangle_{K_{BS}}$ to represents a request from B to authenticate with A through the TTP S. Note that we cannot map the message to a block of the form $\langle \mathsf{Owner}, A, N, K \rangle_{K_{BS}}$ as no session key K is present in the message and K is mandatory for role Owner. In other words, to see a message as an authentication block, we need that every element required by the block is present in the original message. Note also that M is "discarded" in authentication blocks. This reflects the fact that it is irrelevant for entity authentication.

3 Principles for Entity Authentication

The formalization of our principles draws on a trace-based model that we adopt for describing protocol executions. The principles are stated as inference rules that specify (i) the format of the ciphertexts that a trusted party should generate to complete an authentication session, and (ii) the conditions under which such ciphertexts should be generated to provide the intended authentication guarantees.

The ciphertexts can be in two forms: either authentication blocks encrypted with long-term keys (which we assume to be robust) or any other message encrypted with

session keys which are possessed by the trusted parties, and are only circulated among them within authentication blocks.

The inference rules predicate the generation of new messages or actions by each trusted principal on (i) previous actions by the same principal and (ii) on the structure of the ciphertexts that principal received at earlier stages of the protocol. Hence, the rules effectively assume that all the ciphertexts reach *all* parties, included their intended recipients. This may at first be understood as a limitation on the power of the intruder, because it prevents attacks to be mounted based on the interception (i.e., subtraction from the net) of ciphertexts. This is not the case, however, as intercepting a message may only break liveness properties, such as *fairness* in contract-signing, where two parties seek guarantees that none of them will get the contract signed before the other one (see, e.g., [21,25]). Authentication, instead, is (formalized as) a *safety* property, which is immune to attacks based on message interception. We start by formalizing the notion of safety.

3.1 Traces, Markers and Events

We represent (possibly concurrent) protocol runs in terms of traces that collect the sequence of encrypted messages generated during the runs as well as additional *marker* actions that we use to state our safety result. The structure of traces is defined as follows:

$$\begin{array}{ll} \textit{Markers} & \mu ::= run_{\mathsf{R}}(I_1, I_2) \mid commit(I_1, I_2) \mid check(I, N) \\ \textit{Actions} & \alpha ::= \mu \mid I \triangleright \{M\}_K \mid I \triangleright \langle \mathsf{R}, J, N, K \rangle_{K_{I_1 I_2}} \\ \textit{Traces} & \sigma ::= \varepsilon \mid \alpha :: \sigma \end{array}$$

The two actions $I \triangleright \{M\}_K$ and $I \triangleright \langle \mathsf{R}, J, N, K \rangle_{K_{I_1 I_2}}$ represent the generation by entity I of $\{M\}_K$ and $\langle \mathsf{R}, J, N, K \rangle_{K_{I_1 I_2}}$, respectively. As for the markers, they are not part of the protocol runs: we include them in the traces to help define the desired safety property. Specifically, $check(I, N)$ indicates the checking of the nonce N by the entity I. Marker $run_{\mathsf{R}}(I_1, I_2)$ indicates the intention of (equivalently, the start of a protocol run by) I_1 to authenticate itself with I_2, by exploiting the role R. Note that there might be many $run_{\mathsf{R}}(I_1, I_2)$ markers corresponding to many (either parallel or sequential) protocol sessions (but we will require that all have the same role R). We write $run(B, A)$ instead of $run_{\mathsf{R}}(B, A)$ when the role R is immaterial. Finally, $commit(I_1, I_2)$ marks the completion of the authentication session.

We can now define the desired property of entity authentication in terms of a *correspondence* between actions of the participants [23,17], checked on the protocol traces.

Definition 2 (Safety). *A trace σ is safe if and only if whenever $\sigma = \sigma_1 :: commit(B, A) :: \sigma_2$, one has $\sigma_1 = \sigma_1' :: run(A, B) :: \sigma_1''$, and $\sigma_1' :: \sigma_1'' :: \sigma_2$ is safe.*

Intuitively, a trace is safe if every $commit(A, B)$ in the trace is preceded by a corresponding $run(B, A)$ action. Next, we discuss the rules for forming safe traces.

Table 1. Inference Rules

Environments

$$\textsc{Empty}\ \frac{}{\emptyset \vdash \diamond} \qquad \textsc{New}\ \frac{\Gamma \vdash \diamond \quad V \text{ fresh in } \Gamma \quad V \text{ key or nonce}}{\Gamma, new(I,V) \vdash \diamond}$$

Principles

$$\textsc{Env}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \varepsilon} \qquad \textsc{Principle 1}\ \frac{\Gamma \vdash \sigma \quad run_{\mathsf{R}}(A,B) \in \sigma \Rightarrow \mathsf{R} = \mathsf{Claim}}{\Gamma \vdash \sigma :: run_{\mathsf{Claim}}(A,B) :: A \triangleright \langle \mathsf{Claim}, A, N \rangle_{K_{AB}}}$$

$$\textsc{Principle 2}\ \frac{\Gamma \vdash \sigma \quad run_{\mathsf{R}}(A,B) \in \sigma \Rightarrow \mathsf{R} = \mathsf{Verif}}{\Gamma \vdash \sigma :: run_{\mathsf{Verif}}(A,B) :: A \triangleright \langle \mathsf{Verif}, B, N \rangle_{K_{AS}}}$$

$$\textsc{Principle 3}\ \frac{\Gamma, new(S,N) \vdash \diamond \quad \Gamma \vdash \sigma \quad I \triangleright \langle \mathsf{Verif}, B, N \rangle_{K_{AS}} \in \sigma}{\Gamma, new(S,N) \vdash \sigma :: check(S,N) :: S \triangleright \langle \mathsf{Claim}, A, N' \rangle_{K_{BS}}} \qquad \textsc{Principle 4}\ \frac{\Gamma \vdash \sigma \quad I \triangleright \langle \mathsf{Verif}, B, N \rangle_{K_{AS}} \in \sigma}{\Gamma \vdash \sigma :: S \triangleright \langle \mathsf{Claim}, A, N \rangle_{K_{BS}}}$$

$$\textsc{Principle 5}\ \frac{\Gamma, new(S,K) \vdash \diamond \quad \Gamma \vdash \sigma}{\Gamma, new(S,K) \vdash \sigma :: S \triangleright \langle \mathsf{Owner}, A, N_A, K \rangle_{K_{BS}} :: S \triangleright \langle \mathsf{Owner}, B, N_B, K \rangle_{K_{AS}}}$$

$$\textsc{Principle 6}\ \frac{\Gamma \vdash \sigma \quad I \triangleright \langle \mathsf{Owner}, B, N, K \rangle_{K_{AS}} \in \sigma \quad run_{\mathsf{R}}(A,B) \in \sigma \Rightarrow \mathsf{R} = \mathsf{Owner}}{\Gamma \vdash \sigma :: run_{\mathsf{Owner}}(A,B) :: A \triangleright \{M\}_K}$$

Authentication Guarantees

$$\textsc{Authentication 1}\ \frac{\Gamma, new(B,N) \vdash \diamond \quad \Gamma \vdash \sigma \quad J \triangleright \langle \mathsf{Claim}, A, N \rangle_{K_{IB}} \in \sigma \quad I \in \{S, A\}}{\Gamma, new(B,N) \vdash \sigma :: check(B,N) :: commit(B,A)}$$

$$\textsc{Authentication 2}\ \frac{\Gamma, new(B,N) \vdash \diamond \quad \Gamma \vdash \sigma \quad I \triangleright \langle \mathsf{Owner}, A, N, K \rangle_{K_{SB}} \in \sigma \quad J \triangleright \{M\}_K \in \sigma \quad J \neq B}{\Gamma, new(B,N) \vdash \sigma :: check(B,N) :: commit(B,A)}$$

3.2 Principles as Inference Rules

The inference rules, reported in Table 1, derive judgments of the form $\Gamma \vdash \sigma$, where σ is a trace, and Γ is an environment. Environments collect the *events* of the protocols, i.e. the creation of session keys and nonces:

$$\textit{Environments} \quad \Gamma ::= \emptyset \mid \Gamma, new(I,V)$$

where $new(I, V)$ traces the generation of the nonce or key V by the entity I. Environments are used in the inference system to ensure the freshness of nonces and session keys. The intuition is as follows: when a new nonce is generated, or a session key created, it is included in the environment. A nonce (key) can only be generated if it does not occur in the current environment. Nonces can be checked, (keys used) only if they are available in the environment, and once a nonce is checked (a key used) the corresponding event is discarded from the environment, and hence it is never available again for checking (usage).

The purpose of the inference rules is to generate *safe* traces, that is traces that provide the intended authentication guarantees for a protocol session. Indeed, we can prove the following result, stating that protocols designed according to our principles do achieve the desired authentication guarantees.

Theorem 1 (Safety). *If $\Gamma \vdash \sigma$ then σ is safe.*

We now proceed with the description of the deduction rules. The first block of rules derive auxiliary judgments $\Gamma \vdash \diamond$, that decree environments "well-formed": these rules should be self-explanatory: an environment is well-formed just in case it does not contain duplicate nonces and/or keys. The remaining rules, which formalize our principles, have the following uniform structure:

$$\frac{\Gamma \vdash \sigma \quad (\textit{side conditions})}{\Gamma' \vdash \sigma'}$$

where σ' extends σ, and define the conditions under which the judgment in the conclusion may be derived from the judgment in the premise by including further actions in the trace. The side conditions only test the presence of certain ciphertexts, and marker actions, in the current trace σ.

Protocols without TTP (Direct Authentication). Principal A may authenticate with B by using a long-term key she shares with B.

PRINCIPLE 1 *Principal A may start authenticating with principal B by generating an authentication block of the form $\langle \mathsf{Claim}, A, N \rangle_{K_{AB}}$.*

Since this is the start of an authentication session, it is marked by including $run_{\mathsf{Claim}}(A, B)$ in the current trace. Other concurrent runs $run_{\mathsf{R}}(A, B)$ are possible, given that $\mathsf{R} = \mathsf{Claim}$. Note that this principle is consistent with the intended guarantees provided by decrypting a block with Claim role. When B decrypts the block $\langle \mathsf{Claim}, A, N \rangle_{K_{AB}}$, and verifies the nonce N, he is guaranteed that A has initiated an authentication session with him; moreover, if A receives such a message, she will discard it as the block is declaring herself as claimant.

Example 2. To motivate, consider the following protocol inspired to the ISO symmetric-key, two-pass unilateral authentication protocol [18,15]:

$$1)\ B \rightarrow A : N_B \qquad 2)\ A \rightarrow B : \{M, A, N_B\}_{K_{AB}}$$

B, the verifier, sends a freshly generated nonce to A as a challenge. A, the claimant, completes the authentication session by sending a message M, A and N_B encrypted using K_{AB}. This proves to B the identity of the claimant A, as only A and B know K_{AB}. Since identifier A represents the *claimant* of the protocol run, we make this information explicit by mapping message 2 into the authentication block $\langle \mathsf{Claim}, A, N_B \rangle_{K_{AB}}$ (note that M is not in the block as it is unessential for authentication). The protocol is a "good one" as it follows principle 1.

Interestingly, we can show that failing to comply with the previous principle may lead to protocol flaws. In particular, we consider the cases in which either A or N_B are missing in Message 2. In both cases there is no way to represent the message as a valid authentication block as one of the fundamental components is missing. And in both cases the protocol is insecure. Suppose first that A is missing: Message 2 becomes $\{M, N_B\}_{K_{AB}}$. Observing message $\{M, N\}_{K_{AB}}$, there is no way to tell who between A and B is the claimant, so it is impossible to distinguish between messages that belong to a protocol run and messages that belong to another concurrent protocol run with reversed roles. This fact can be used effectively by an adversary as follows:

$$\begin{array}{l} \text{1.a)}\ B \rightarrow E(A) : N_B \\ \text{1.b)}\ E(A) \rightarrow B : N_B \\ \text{2.b)}\ B \rightarrow E(A) : \{M, N_B\}_{K_{AB}} \\ \text{2.a)}\ E(A) \rightarrow B : \{M, N_B\}_{K_{AB}} \end{array}$$

With such a "reflection attack" an enemy can trick B into believing that A has been actively involved in the protocol run "a", while in fact it has not. As a further example, consider a version of the protocol that disregards the nonce, i.e., one in which message 2) is replaced by the message $\{A, M\}_{K_{AB}}$. Here the problem is that there is no way for B to tell if the message belongs to the current run or to a previous one, so an adversary can convince B that it is talking to A without A actively participating:

$$\text{old)}\ A \rightarrow B : \{A, M\}_{K_{AB}} \quad \ldots\ldots \quad \text{new)}\ E(A) \rightarrow B : \{A, M\}_{K_{AB}}$$

Protocols with TTP (Indirect Authentication). When a TTP is present, Alice may authenticate to Bob using the TTP as a mediator. In this setting, we assume that the only way for Alice and Bob to communicate is via the TTP, with which they share a long-term key. The role Verif aims at dealing with this kind of indirect authentication.

The second principle is similar to the previous one, only, it marks the start of the authentication session with $run_{\mathsf{Verif}}(A, B)$ and represents the start of an authentication session through a TTP.

PRINCIPLE 2 (Authentication through TTP) *Principal A may start authenticating herself with principal B by generating an authentication block $\langle \mathsf{Verif}, B, N \rangle_{K_{AS}}$.*

Note that this principle is consistent with the intended semantics of Verif role: it allows A to communicate to a TTP her intention of authenticating with B. This principle is useful when combined with other principles that allow the TTP to forward authentication to the intended verifier B. Notice that, as in the previous principle, A may be running other concurrent authentication sessions, provided that they are all marked with run_{Verif}.

PRINCIPLE 3 (Verified authentication forwarding) *TTP S may declare to a principal B that a principal A has recently started an authentication session with B, if S has decrypted a block $\langle \mathsf{Verif}, B, N \rangle_{K_{AS}}$ and checked the validity of N. To do so, S generates an authentication block $\langle \mathsf{Claim}, A, N' \rangle_{K_{BS}}$.*

The principle above states that a TTP can forward authentication from A to B if the TTP is guaranteed that A has recently initiated an authentication session with B. This guarantee is provided by the decryption (and the nonce checking) of the authentication block $\langle \mathsf{Verif}, B, N \rangle_{K_{AS}}$, and this principle is thus consistent with the intended semantics of Claim and Verif roles.

Notice that to guarantee that N is never used again in the execution, PRINCIPLE 3 discards the event $new(S, N)$ from Γ.

Example 3 (nonce-based Wide Mouth Frog Protocol [1]). The Wide Mouth Frog Protocol [5] achieves unilateral authentication using a TTP. The original version is based on timestamps, here we consider the nonce-based version presented in [12].

1) $A \rightarrow S : A$
2) $S \rightarrow A : N_S$
3) $A \rightarrow S : A, \{B, K_{AB}, N_S\}_{K_{AS}}$
4) $S \rightarrow B : *$
5) $B \rightarrow S : N_B$
6) $S \rightarrow B : \{A, K_{AB}, N_B\}_{K_{BS}}$

This protocol can be seen as a direct application of principles 2 and 3. As a matter of fact, the two encrypted messages 3 and 6, can be abstracted as the two authentication blocks $\langle \mathsf{Verif}, B, N_S \rangle_{K_{AS}}$ and $\langle \mathsf{Claim}, A, N_B \rangle_{K_{BS}}$, respectively. The latter block provides an authentication guarantee to B. Note that the two encrypted messages have the same form. Assigning them to two different authentication blocks make their interpretation very different and allows us to easily reason about which guarantees the two messages provide.

The next principle captures a variant of authentication forwarding that allows the TTP to forward authentication even if it has not verified the claimant's nonce. The idea is that the nonce sent by the claimant is forwarded to the verifier who is in charge of checking its validity.

PRINCIPLE 4 (Non-verified authentication forwarding) *TTP S may declare to a principal B that a principal A has started an authentication session with B, if S has decrypted a block $\langle \mathsf{Verif}, B, N \rangle_{K_{AS}}$. To do so, S generates an authentication block of the form $\langle \mathsf{Claim}, A, N \rangle_{K_{BS}}$.*

The difference with respect to principle 3 is that N is not checked by the TTP but is forwarded to B, in order to let him check its validity. This principle is again consistent with the intended semantics of authentication blocks: the TTP is assured that A has started authenticating with B even though it does not know anything about the freshness of this intention; for this reason, the TTP forwards the nonce N to B so that B may check it and verify that the authentication session started by A is also recent.

[1] This protocol has a *type-flaw* attack (cf. [12,2,3]) which we disregard, because of the perfect cryptography assumption.

A possible scenario is the following: the nonce is originated by the verifier, sent to the claimant who inserts it into the authentication request to the TTP; finally the nonce is forwarded back to the verifier for the final check. This is what the "Woo and Lam Shared-Key Protocol" does [24].

Example 4 (Flaws in the original Woo and Lam Protocol). Here we report the original (flawed) version of the protocol [24].

$$\begin{array}{ll} 1)\ A \rightarrow B : A & 4)\ B \rightarrow S : \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}} \\ 2)\ B \rightarrow A : N_B & 5)\ S \rightarrow B : \{N_B\}_{K_{BS}} \\ 3)\ A \rightarrow B : \{N_B\}_{K_{AS}} & \end{array}$$

Message 3 is encrypted with the key shared with S and is thus directed to S (even if it is sent to B who is in charge of forwarding it to S inside message 4). It represents the declaration of the intention of A to authenticate with B. However, note that A does not include in the ciphertext of message 3 the label of B. Thus, it is not possible to apply principle 2. The same happens in the last message, where S does not include any identity label. This makes Principle 4 inapplicable. It is interesting to remark that the corrected version of this protocol [12] uses the identifiers exactly as required by our principles.

Protocols with TTP Based on Session Keys. Another way for Alice to authenticate with Bob through a TTP is to establish a fresh session key with Bob and then use it to authenticate her. PRINCIPLE 5 states how key distribution should happen:

PRINCIPLE 5 (Session key distribution) *TTP S may distribute to principals A and B a freshly generated key K by generating two authentication blocks $\langle \mathsf{Owner}, B, N, K \rangle_{K_{AS}}$ and $\langle \mathsf{Owner}, A, N', K \rangle_{K_{BS}}$.*

The principle above shows how the Owner role is used: the TTP communicates to A (B) that B (A) owns the key K. Note that, in order to guarantee the secrecy of the session key, we are assuming that the TTP will never send other messages containing K. As in Table 1, the event $new(S, K)$ is discarded from Γ to ensure that K is never used again.

The next principle allows A to send to B a message encrypted with a session key K.

PRINCIPLE 6 (Authentication through session-keys) *Principal A may start an authentication session with B, by generating a ciphertext $\{M\}_{K_s}$, provided that A has decrypted a block $\langle \mathsf{Owner}, B, N, K \rangle_{K_{AS}}$ and checked the validity of N.*

The condition $I \triangleright \langle \mathsf{Owner}, B, N, K \rangle_{K_{AS}} \in \sigma$, binds K to the principal which shares K. Applying this principle, B may conclude that A has initiated an authentication session with him, if he receives a message $\{M\}_{K_s}$ and knows that (i) K is a fresh key owned by A and (ii) he did not generate $\{M\}_{K_s}$. An example of a protocol based on this kind of authentication is the Amended Needham Schroeder Shared-Key Protocol [19].

Authentication Guarantees. In view of the principles we have outlined, there are two situations that can drive an entity to accept an authentication request. An entity B accepts the authentication request of A whenever one of the following two conditions holds:

AUTHENTICATION 1 either B receives and checks (the nonce of) an authentication block $\langle \mathsf{Claim}, A, N \rangle_{K_{IB}}$, or $\langle \mathsf{Claim}, A, N, K \rangle_{K_{IB}}$, where I is either A or a TTP identifier;

AUTHENTICATION 2 or, B has received and checked (the nonce of) an authentication block in the form $\langle \mathsf{Owner}, A, N, K_s \rangle_{K_{SB}}$, with S a TTP, and receives a message $\{M\}_K$ that he did not generate.

In condition AUTHENTICATION 1, if I is A, then the authentication request is *directly* made by A. Otherwise, I must be a TTP, say S, and the request is indirectly made S on behalf of A. A further remark is in order for the side-condition $J \neq B$ in AUTHENTICATION 2 . Here we are assuming that B, which is *committing* (hence completing the authentication session) has the ability to tell that he is not the originator of the ciphertext $\{M\}_K$ (see, e.g., the Amended Needham Schroeder Shared-Key Protocol [19]).

4 Conclusion and Related Work

We have given an in-depth analysis of the roles that message components play in authentication protocols, and we have investigated how a certain component contributes to the task of achieving entity authentication. We identified three simple roles, namely *Claimant*, *Intended verifier* and *Key owner*, that allow us to extract general principles to be followed to protect protocols against attacks. These principles are formalized in terms of rules for protocol parties and lead us to prove that any protocol following these rules (i.e., designed according to the principles) achieves entity authentication. Our approach should be understood as a first step of the development a new analysis technique for authentication protocols. Our goal is to develop (possibly automated) static analysis of authentication protocols based on the principles explained in this paper. At the present stage, we are able to judge if a certain trace is obtained according to the principles: if this is the case, we are guaranteed that such a trace is safe with respect to a specific notion of entity authentication. The next step is to have a language for specifying cryptographic protocols and a technique for proving that a certain protocol specification is designed according to the principles. This would imply that all the execution traces of the protocol are safe and, consequently, that the protocol guarantees entity authentication. We are presently studying how to obtain this on a language similar to spi calculus [1].

The formal model we have developed appears to be fairly general, but a number of extensions would be desirable to capture a wider class of protocols. We are currently studying how to extend the model in order to manage asymmetric and nested encryption, and in order to capture other important security properties such as message authentication and key distribution.

The idea of associating to principals the roles played in a cryptographic protocol is not completely new. Drawing on earlier work by Bieber in [4], in [22] Snekkenes develops a logic for the analysis of cryptographic protocols, to detect multi-role flaws, i.e., flaws that may result from a principal taking on multiple roles during different sessions of a protocol (an instance of this situation is the reflection attack of Example 3, where A plays both the initiator and responder roles). Our idea of role is however appreciably

different. In particular, we focus on the role that principals play in an authentication task (e.g. claimant or verifier) instead of considering usual protocol roles like initiator and responder. Notice that reasoning on usual protocol roles does not always give information about "who is willing to authenticate with who". For example in mutual authentication protocols both initiator and responder act as claimant and verifier simultaneously, since in such protocols there are two authentication tasks (the authentication of A to B and vice versa).

In [12,11] Gordon and Jeffrey provide the spi calculus with a type system for analyzing authentication protocols based on symmetric and asymmetric encryption, respectively. These papers are very related to our work. First, their definition of safety exploits, as we do, the Woo and Lam idea of correspondence assertions [23]. Moreover, as stated above, our final aim is to obtain a static analysis of cryptographic protocols as they already do. There are however several important differences. The underlying idea of using roles and principles for generating authentication blocks is new and seems to be promising for obtaining simpler static analyses. Indeed, type definitions are the only human task required in [12,11], but types are quite complex, as the type of nonces keeps all the information regarding authentication tasks between principals. This implies dependent types and forces the use of casting inside processes. Our approach is completely different. We focus on the guarantees that the reception of a given ciphertext, as a whole, gives to a principal and we do not reason about single components. For instance, in our model a nonce does not, by itself, convey any information regarding principals. It is just used for ensuring freshness to the enclosing authentication block. We expect that a static analysis based on these ideas will be simpler as the only human effort required is the binding of every identity label present in a ciphertext to one of the three roles.

Further related work, which aims at reasoning about authentication guarantees provided by messages, is the one based on belief logics, like, e.g., BAN [5], GNY [10]. The main difference with such an approach is that we do not have any kind of protocol idealization. Indeed, we reason about "concrete" execution traces, directly connected to the structure of ciphertexts. An interesting paper that goes in the direction of eliminating the idealization step of the approaches above is [7] by Durgin, Mitchell and Pavlovic. They propose a logic for proving security properties, which is designed around a process calculus derived from Strand Spaces. The calculus has a semantics based on sets of traces and deals with asymmetric encryption. Our model is not directly comparable with the one proposed in the above mentioned paper, since we consider symmetric encryption and rely on a different formalization of authentication properties. Further work needs to be put in place to gain a deeper understanding of the relationships between the two approaches.

References

1. M. Abadi and A. D. Gordon. "A Calculus for Cryptographic Protocols: The Spi Calculus". *Information and Computation*, 148(1):1–70, 1999.
2. M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.
3. R. Anderson and R. Needham. Programming satan's computer. In Jan van Leeuwen, editor, *Computer Science Today — Recent Trends and Developments*, volume 1000 of *ln-cs*, pages 426–440. 1995.

4. P. Bibier. A logic of communication in a hostile environment, 1990. Proceedings the Computer Security Foundations Workshop III. IEEE Computer Society Press, 1422.
5. M. Burrows, M. Abadi, and R. Needham. "A Logic of Authentication". *Proceedings of the Royal Society of London*, 426(1871):233–271, 1989.
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. N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols, 2001. 14-th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, June 11–13.
8. 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.
9. D. Gollmann. "What do we mean by Entity Authentication". In *Proceedings of the 1996 Symposium on Security and Privacy*, pages 46–54. IEEE Computer Society Press, 1996.
10. L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.
11. A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. In *15th IEEE Computer Security Foundations Workshop — CSFW'01*, pages 77–91, Cape Breton, Canada, 24–26 June 2002. IEEE Computer Society Press.
12. A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In 14th IEEE Computer Security Foundations Workshop (CSFW-14),pages 145–159, June 2001.
13. J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *13th IEEE Computer Security Foundations Workshop — CSFW'00*, pages 255–268, Cambridge, UK, 3–5 July 2000. IEEE Computer Society Press.
14. ISO/IEC. *Information Technology-Security Tecniques-Entity Authentication Mechanisms, Part 1:General Model*. 1991.
15. ISO/IEC. *Information Technology-Security Tecniques-Entity Authentication Mechanisms, Part 2:Entity Authentication using Simmetric Tecniques*. 1993.
16. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.
17. G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.
18. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
19. R. M. Needham and M. D. Schroeder. Authentication revisited. *ACM SIGOPS Operating Systems Review*, 21(1):7–7, 1987.
20. Lawrence C. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.
21. S. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of CSFW'98*, pages 54–65. IEEE Press, 1998.
22. E. Snekkenes. Roles in cryptographic protocols, 1992. In Proceedings of the 1992 IEEE Symposium on Security and Privacy, pages 105-119. IEEE Computer Society Press.
23. T.Y.C. Woo and S.S. Lam. "A Semantic Model for Authentication Protocols". In *Proceedings of 1993 IEEE Symposium on Security and Privacy*, pages 178–194, 1993.
24. T. Y. C. Woo and S. S. Lam. Authentication for distributed systems, from computer, january, 1992. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1992*. 1992.
25. J. Zhou and D. Gollman. A fair non-repudiation protocol. In *Proc. of Symposium in Research in Security and Privacy*, pages 55–61. IEEE Press, 1996.