Injective Synchronisation: an extension of the authentication hierarchy

C.J.F. Cremers [a], S. Mauw [a], E.P. de Vink [a,b]

[a] *Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

[b] *LIACS, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands*

Abstract

Authentication is one of the foremost goals of many security protocols. It is most often formalised as a form of agreement, which expresses that the communicating partners agree on the values of a number of variables. In this paper we formalise and study an intensional form of authentication which we call *synchronisation*. Synchronisation expresses that the messages are transmitted exactly as prescribed by the protocol description. Synchronisation is a strictly stronger property than agreement for the standard intruder model, because it can be used to detect *preplay attacks*. In order to prevent *replay attacks* on simple protocols, we also define *injective synchronisation*. Given a synchronising protocol, we show that a sufficient syntactic criterion exists that guarantees that the protocol is injective as well.

Key words: Security Protocols, Authentication, Agreement, Injectivity

1 Introduction

The security property studied the most in the field of security protocol analysis is *authentication*. However, contrary to the requirement of *secrecy*, there is no general consensus on the meaning of authentication. In fact, as indicated by Lowe [22], there is a hierarchy of authentication properties, the most popular of which is *agreement*. Agreement means that two parties involved in a protocol are guaranteed to agree upon the values of variables after successful completion of the protocol.

Email addresses: `ccremers@win.tue.nl` (C.J.F. Cremers), `sjouke@win.tue.nl` (S. Mauw), `evink@win.tue.nl` (E.P. de Vink).

Using terminology from Roscoe [27] agreement is a so-called *extensional* security property, which means that it takes into account the effect the protocol achieves. Roscoe observes that it is often hard to provide an extensional specification that exactly captures the requirements of the protocol and detects all relevant attacks. In contrast to extensional characterisations, *intensional* characterisations of security properties are induced by the form or structure of the protocol. Roscoe introduces the notion of *canonical intensional specification*, which expresses that "no node can believe a protocol run has completed unless a correct series of messages has occurred (consistent as to all the various parameters) up to and including the last message the given node communicates." So, it states that all parties involved in a protocol, after completion of their role, are convinced that the protocol has been executed according to its rules.

Roscoe clarifies his description of intensional specification by providing a sample implementation in CSP. However, to precisely understand the nature of intensional specifications, a fundamental approach is beneficial. Therefore, we formally define the notion of *synchronisation*, which is an intensional authentication property requiring that all protocol messages occur in the expected order with the expected values.

Important to the approach here is that we consider *local synchronisation claims*. That means that an agent may decide that the complete protocol has been executed exactly as expected, based on his local observations only. These observations only take into account the contents of the communications that the agent was involved in. Whenever such an agent successfully completes a run of a synchronising protocol, all other parties involved in the protocol have executed their part exactly as expected.

Thus, for a two-party protocol, we have the following informal definition of synchronisation:

> Initiator I considers a protocol synchronising, whenever I as initiator completes a run of the protocol with responder R, then R as responder has been running the protocol with I. Moreover, all messages are received exactly as they were sent, in the order as described by the protocol.

This definition extends in a natural way to multi-party protocols.

It is well-known that such an authentication property may give rise to replay attacks. Consider, e.g., an electronic car key that sends an encrypted signal to unlock a car. From the view point of the car, this protocol synchronises, since an intruder cannot construct the encrypted unlock message himself, so it must have been sent by the car key. However, once the intruder has been able to eavesdrop on the communication between the car key and the car, he can replay the unlock message and open the car at will. In general, such protocols are ruled out by requiring *injective* authentication. For the car key example, it implies that for each time that the car receives an unlock message there must have been a unique unlock event of the car key. Injectivity is a property that is not captured in Roscoe's definition of *canonical intensional specifications*. In order to fill this gap, we introduce *injective synchronisation*, which, in case of a two-party protocol informally states the following:

> Initiator I considers a protocol injectively synchronising if the protocol synchronises and each run of I corresponds to a unique run of R.

Most often, security properties, such as agreement, are first defined at the conceptual level and then within a specific security model (cf. the CSP model in [28]). For our definition of injective synchronisation we will provide as general a framework as possible. For instance, we will abstract from the precise contents of the protocol messages, because we only have to reason about their correspondence. This makes it possible to easily interpret our results in different security protocol semantics, such as Strand Spaces [31] and Casper/FDR [28], for example.

The framework that we define also allows us to express injective agreement over all variables in an intensional way, which makes a formal comparison possible. Injective synchronisation can be shown to be strictly stronger than injective agreement in the standard intruder model, and thus can be added as top element of Lowe's authentication hierarchy. The rather subtle difference between the two can be best explained using so called *preplay attacks* [24]. Such an attack occurs if an attacker is able to predict a protocol message and inject it into the system before it was actually created by the intended sender. Synchronising protocols are not susceptible to preplay attacks, while agreeing protocols possibly are. Preplay attacks are not very common, and indeed most protocols from the Spore library [30] that satisfy injective agreement, satisfy injective synchronisation as well.

An application of our treatment is found in the verification of injectivity. In practice, many verification tools are based on model checking and make use of a *counting argument* to verify that a protocol is injective. This will, in general, only provide an approximation of injectivity. Many other approaches, with some notable exceptions, do not seem to pay much attention to injectivity. Our formalisation of injective synchronisation allows us to describe a syntactic property which is sufficient for deciding whether a synchronising protocol is injective. By syntactic, we mean that it can be directly derived from the syntactic description of the security protocol.

The above mentioned goals lead to the following structure of the present paper. In Section 2 below we gather the machinery required for our description of security protocols and provide a formal definition of injective synchronisation.

The relation with injective agreement is described in Section 3. We also explain how Lowe's authentication hierarchy is extended. The question of how to verify injective synchronisation is discussed in Section 4. Here, the *loop* property is introduced as a syntactic criterion for verifying injectivity of a synchronising protocol. In Section 5 we discuss related literature and we close off with some concluding remarks in Section 6.

Acknowledgements. We would like to thank the anonymous reviewers whose helpful comments have improved this paper.

2 Model and definition of synchronisation

In this section we introduce the notions of non-injective synchronisation and injective synchronisation. Synchronisation is based on the property that every successful execution of a protocol by an agent implies that its communication partners exactly follow their roles in the protocol and exchange the intended messages in the intended order. First, we provide a formal definition of a security protocol and the order implied on its events, referred to as protocol events. Next, we postulate a trace semantics involving traces of trace events and introduce the concept of a cast that captures role instantiations or runs. Finally, we provide the definition of the two versions of synchronisation. The three-message version of the well-known Needham-Schroeder protocol [25] is used as a running example.

Definition 1 (Abstract Security Protocol) *An* abstract security proto*col P over the set Role is a tuple $\langle\, PE,\ role,\ (\prec_{\mathsf{R}})_{\mathsf{R}\in Role},\ \rightsquigarrow\, \rangle$, where*

(i) the set PE of protocol events is a disjoint union

$$PE = SendPE + ReadPE + ClaimPE$$

of the set $SendPE$ of send protocol events, the set $ReadPE$ of read protocol events, and the set $ClaimPE$ of claim protocol events;

(ii) the mapping $role\colon PE \rightarrow Role$ is the so-called role assignment function of P;

(iii) for each role $\mathsf{R} \in Role$, the relation $\prec_{\mathsf{R}}$ is a strict ordering on the set $role^{-1}(\mathsf{R})$;

(iv) the communication relation $\rightsquigarrow$ of P is a 1–1 correspondence between $SendPE$ and $ReadPE$, i.e. a bijective map $\rightsquigarrow\colon SendPE \rightarrow ReadPE$.

The protocol order $\preccurlyeq_P$ of P is defined by

$$\preccurlyeq_P = \Big(\bigcup_{\mathsf{R}\in Role} \prec_{\mathsf{R}} \cup \rightsquigarrow \Big)^{*},$$

i.e., $\preccurlyeq_P$ *is the least reflexive and transitive ordering on PE respecting each role order* $\prec_{\mathsf{R}}$ *as well as the communication relation* $\rightsquigarrow$*. The notation ASP denotes the collection of all abstract security protocols.*

The set PE contains, at the level of the protocol description, all the actions to be performed in the protocol. We distinguish protocol events for sending, reading and claiming. Although one may include other protocol events as well, e.g. to model internal activity within a role, we do not do so here; the distinction of send protocol events, read protocol events, and claim events, suffices for our purposes below.

We assume that each protocol event belongs to a single role; some syntactic sugar can be used to distinguish similar protocol events that belong to different roles. A protocol event is bound to a role via the role description function $\mathit{role}\colon PE \to \mathit{Role}$. We impose a (sequential) structure on the set of protocol events belonging to a role R by means of the total ordering $\prec_{\mathsf{R}}$. Thus, for any role $\mathsf{R} \in \mathit{Role}$ and $\varepsilon_1, \varepsilon_2 \in PE$ such that $\mathit{role}(\varepsilon_1) = \mathsf{R}$ and $\mathit{role}(\varepsilon_2) = \mathsf{R}$ we have that

$$\varepsilon_1 \prec_{\mathsf{R}} \varepsilon_2 \ \vee \ \varepsilon_1 = \varepsilon_2 \ \vee \ \varepsilon_1 \succ_{\mathsf{R}} \varepsilon_2.$$

In fact, we consider an abstract security protocol P as a collection of communicating sequential processes. Each of the sequential components is carried by a specific role. The communication is governed by the communication relation $\rightsquigarrow$. This relation prescribes how send protocol events and read protocol events match. We require that for each send protocol event σ there is a unique read protocol event ϱ such that $\sigma \rightsquigarrow \varrho$, and, vice versa, that for each read protocol event ϱ there is a unique send protocol event σ such that $\sigma \rightsquigarrow \varrho$. No unmatched send protocol events, nor unmatched read protocol events are allowed in the abstract protocol P.

Of course, at the level of the trace semantics of P as introduced below, a strict matching as given by the communication relation $\rightsquigarrow$ is not required. For example, in a Dolev-Yao setting where the intruder is capable of intercepting messages, unmatched sends may occur in a trace. However, we stress that at the level of an abstract security protocol, its roles and protocol events, the intruder is not involved. Put otherwise, the abstract security protocol describes what is intended to happen.

The orderings $\prec_{\mathsf{R}}$ on the sets $\mathit{role}^{-1}(\mathsf{R})$ and the communication relation $\rightsquigarrow$ induce naturally an ordering $\preccurlyeq_P$ across the roles, on the complete set PE of protocol events. For example, a protocol event ε in a role R is considered to occur before a protocol event ε' in a role R' in case there is a send event σ of R and a read protocol event ϱ of R' such that

$$\varepsilon \prec_{\mathsf{R}} \sigma \wedge \sigma \rightsquigarrow \varrho \wedge \varrho \prec_{\mathsf{R}'} \varepsilon'.$$

Intuitively, one may require $\preccurlyeq_P$ to be a partial order, excluding cycles such as $\varepsilon \preccurlyeq_P \varepsilon' \wedge \varepsilon' \preccurlyeq_P \varepsilon$, for different protocol events $\varepsilon, \varepsilon'$. As we do not encounter this in the remainder, the relation $\preccurlyeq_P$ is only required to be a preorder.

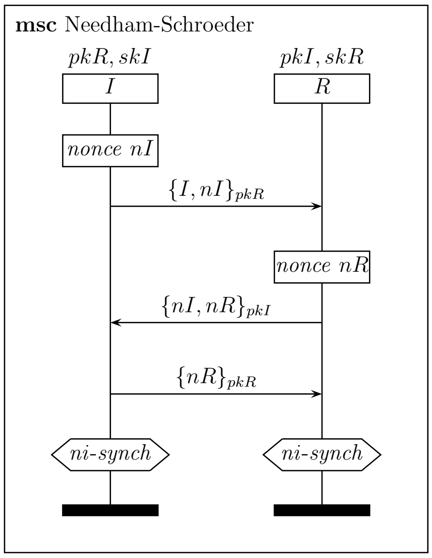

Fig. 1. The NS protocol

We illustrate Definition 1 with the short version of the Needham-Schroeder protocol, referred to as *NS*. Throughout this paper we use Message Sequence Charts (MSC) to illustrate security protocols and attacks. MSC is an ITU-standardized protocol specification language [17]. Figure 1 contains an MSC of the *NS* protocol. The initiator I holds her own secret key skI and the public key pkR of the responder R. Symmetrically, the responder R possesses his own secret key skR and the public key pkI of the initiator I. The initiator first creates a new nonce nI, sends her name I together with the nonce nI, protected by the public key pkR, to the responder. After receipt of this, the responder generates a new nonce nR and sends it, together with the earlier nonce nI, covered by the public key pkI to the initiator. She, in turn, unpacks the message and returns the nonce nR of the responder, encrypted with his public key. Both the initiator and the responder claim that the authentication property *ni-synch*, explained below, holds.

We have $\mathit{Role} = \{\, I,\ R \,\}$ as the set of roles with initiator I and responder R, and for the set of protocol events we put

$$\begin{aligned} PE = \{\, & \mathit{send}_{I\to R}(\{I, nI\}_{pkR}),\ \mathit{read}_{I\to R}(\{I, nI\}_{pkR}), \\ & \mathit{send}_{R\to I}(\{nI, nR\}_{pkI}),\ \mathit{read}_{R\to I}(\{nI, nR\}_{pkI}), \\ & \mathit{send}_{I\to R}(\{nR\}_{pkR}),\ \mathit{read}_{I\to R}(\{nR\}_{pkR}), \\ & \mathit{claim}_I(\mathit{ni\text{-}synch}),\ \mathit{claim}_R(\mathit{ni\text{-}synch}) \,\} \end{aligned}$$

with the following role assignment

$$
\begin{aligned}
\mathit{role}^{-1}(I) = \{\ & \mathit{send}_{I\to R}(\{I, nI\}_{pkR}), \\
& \mathit{read}_{R\to I}(\{nI, nR\}_{pkI}), \mathit{send}_{I\to R}(\{nR\}_{pkR}), \mathit{claim}_I(\mathit{ni\text{-}synch})\ \} \\
\mathit{role}^{-1}(R) = \{\ & \mathit{read}_{I\to R}(\{I, nI\}_{pkR}), \\
& \mathit{send}_{R\to I}(\{nI, nR\}_{pkI}), \mathit{read}_{I\to R}(\{nR\}_{pkR}), \mathit{claim}_R(\mathit{ni\text{-}synch})\ \}.
\end{aligned}
$$

Although some syntactical structure is suggested in the choice of protocol events, such as the subscripts of sender and reader, the payload of a message, including role names and nonces, protected by a public key, this is for presentational purposes only. A less informative selection of protocol events names, say σ_1, σ_2, σ_3, ϱ_1, ϱ_2, ϱ_3, γ_1 and γ_2, or similar, could have been used as well. The choice of claim protocol events $\mathit{claim}_I(\mathit{ni\text{-}synch})$ and $\mathit{claim}_R(\mathit{ni\text{-}synch})$ containing the cryptic *ni-synch* is used, because the protocol serves as a vehicle to illustrate the notion of non-injective synchronisation, abbreviated by *ni-synch*, later.

The role orderings $\prec_I$ and $\prec_R$ on the roles I and R, respectively, are as follows:

$$
\begin{aligned}
\mathit{send}_{I\to R}(\{I, nI\}_{pkR}) &\prec_I \mathit{read}_{R\to I}(\{nI, nR\}_{pkI}) \\
&\prec_I \mathit{send}_{I\to R}(\{nR\}_{pkR}) \prec_I \mathit{claim}_I(\mathit{ni\text{-}synch}) \\
\mathit{read}_{I\to R}(\{I, nI\}_{pkR}) &\prec_R \mathit{send}_{R\to I}(\{nI, nR\}_{pkI}) \\
&\prec_R \mathit{read}_{I\to R}(\{nR\}_{pkR}) \prec_R \mathit{claim}_R(\mathit{ni\text{-}synch})
\end{aligned}
$$

The communication order $\leadsto$ is given as:

$$
\begin{aligned}
\mathit{send}_{I\to R}(\{I, nI\}_{pkR}) &\leadsto \mathit{read}_{I\to R}(\{I, nI\}_{pkR}) && (1) \\
\mathit{send}_{R\to I}(\{nI, nR\}_{pkI}) &\leadsto \mathit{read}_{R\to I}(\{nI, nR\}_{pkI}) && (2) \\
\mathit{send}_{I\to R}(\{nR\}_{pkR}) &\leadsto \mathit{read}_{I\to R}(\{nR\}_{pkR}). && (3)
\end{aligned}
$$

Note that, on the one hand, we have for the protocol order $\preccurlyeq_P$ of NS that

$$
\mathit{send}_{I\to R}(\{nR\}_{pkR}) \preccurlyeq_P \mathit{read}_{I\to R}(\{nR\}_{pkR}) \preccurlyeq_P \mathit{claim}_R(\mathit{ni\text{-}synch})
$$

but, on the other hand,

$$
\mathit{send}_{I\to R}(\{nR\}_{pkR}) \preccurlyeq_P \mathit{read}_{I\to R}(\{nR\}_{pkR}) \not\preccurlyeq_P \mathit{claim}_I(\mathit{ni\text{-}synch})
$$

Such a situation is relevant to the notion of synchronisation. For the claim $\mathit{claim}_I(\mathit{ni\text{-}synch})$ of the initiator role there are only two preceding pairs of send and read events, viz. those of (1) and (2). For the claim $\mathit{claim}_R(\mathit{ni\text{-}synch})$ there are three preceding communication pairs, viz. those of (1), (2) and (3).

We continue the discussion of the Needham-Schroeder example later, where we will see that the authentication property of synchronisation holds with respect to the claim $claim_I(\textit{ni-synch})$ but not for the claim $claim_R(\textit{ni-synch})$.

For the semantical interpretation of an abstract security protocol, we assume that some trace semantics Tr is given. It should be noted that this requirement can be relaxed; also other type of semantics can be used instead. For the sake of presentation we stick to the trace model and refer to the conclusions for a further discussion on this. Let TE be a set of so-called trace events. As for the collection of protocol events, we assume that the set TE is a disjoint union

$$TE = SendTE + ReadTE + ClaimTE + IntrTE.$$

The sets $SendTE$, $ReadTE$, and $ClaimTE$ represent trace events for sending, reading and claiming belonging to normal role behaviour. As internal activity is not taken into account at the protocol level, there are no trace events reflecting this. The elements of the set $IntrTE$, though, have no counterpart in the set PE. These events, called intruder trace events, allow us to represent intruder activity.

We postulate a semantical mapping $Tr\colon ASP \to \mathcal{P}(TE^*)$, for the collection ASP of abstract security protocols, mapping an abstract security protocol to a finite or infinite subset of finite strings of trace events. For technical convenience it is assumed that, for an abstract security protocol P, a trace event occurs at most once in a trace $t \in Tr(P)$. The precise semantics $Tr(P)$ of a protocol P is not relevant for our treatment of authentication here, and is left implicit. However, in order to make the coupling of trace events in TE and protocol events in PE explicit, we assume a partial function

$$pe\colon TE \nrightarrow PE$$

that is defined on the subsets $SendTE$, $ReadTE$ and $ClaimTE$, but not on the set $IntrTE$ of trace events representing malicious activity of the intruder. (Here, $f\colon X \nrightarrow Y$ denotes a partial function f from a set X to a set Y.) The mapping pe extends to traces, i.e. sequences of trace events, in a canonical way: $pe(\epsilon) = \epsilon$ for ϵ the empty sequence in TE^* and PE^*, respectively, $pe(e \cdot t) = pe(e) \cdot pe(t)$ if $e \in TE \backslash IntrTE$, and $pe(e \cdot t) = pe(t)$ if $e \in IntrTE$. The mapping pe enables us to identify the protocol event to which a trace event corresponds.

Traces consist of complete or partial role executions. A role can be executed multiple times, and in different ways, which we call the instantiations of the role. A specific role instantiation is called a run. We assume that there is some mechanism to distinguish trace events that belong to different runs, and we formalise this by partitioning the set of trace events. For this purpose, we assume an equivalence relation π on $TE \backslash IntrTE$, that groups the collection of non-intruder trace events into runs. We use the notation $e_1 \sim_\pi e_2$, for trace events $e_1, e_2 \notin \mathit{IntrTE}$, if these trace events belong to the same equivalence class of π. Similarly, for $e \in \mathit{TE} \backslash \mathit{IntrTE}$, $[e]_\pi$ denotes the equivalence class of π containing the trace event e. Thus, $[e]_\pi$ is the set of all events in the run, or role instance, that contains e. Because a run is an instantiation of a single role, we have that $e \sim_\pi e_2$ implies $\mathit{role}(\mathit{pe}(e_1)) = \mathit{role}(\mathit{pe}(e_2))$.

Furthermore, we assume a mapping

$$\mathit{cont}\colon \mathit{SendTE} \cup \mathit{ReadTE} \to \mathit{Content},$$

that determines the payload of a send or read event in a trace. The specific purpose of *cont* is to establish whether a send and a read event match regarding the content that was sent and received.

We assume in the model that it is possible that the intruder has compromised a number of agents, e.g. has learnt their private key. In our examples we will use one such agent E whose private key skE is known to the intruder. This has a consequence for the evaluation of claims: it is possible that an agent A tries to communicate with the agent E. In this case, the intruder can of course complete the protocol without E ever being present. If an agent A makes a claim c about the protocol whilst communicating with E, it will always fail. We call a claim occurring in a communication with an compromised agent invalid. Of interest is the case in which an agent tries to communicate with agents that are not compromised by the intruder. We express this distinction with the predicate *Valid* on claim trace events, that is true if and only if the claim involves uncompromised agents only. In the typical case, we have that $\mathit{Valid}(c)$ holds if the intended communication partners do not include E, as we will see in the example.

Typically, trace events are given by a more detailed syntax than protocol events are. For example, while distinguishing sends, reads and claims, one might feel the need to attach senders and readers to the send and read event, resulting in constructs as $\mathrm{send}_{A \to B}$ and $\mathrm{read}_{A \to B}$. Such communications may further have some content involving identities like A, B and E or nonces like *123* and *456*. A construct as $\{x\}_{pkA}$ represents, as usual, that x is encrypted by the public key of party A. Below, the distinction between role events like $\mathit{send}_{I \to R}$ and $\mathit{read}_{I \to R}$, on the one hand, and trace events like $\mathrm{send}_{A \to B}$ and $\mathrm{read}_{A \to B}$, on the other hand, is stressed in the notation.

For the Needham-Schroeder protocol discussed above, the trace representing the well-known Lowe attack [20] can have the form

$$\begin{aligned}
&\mathrm{send}_{A\to E}(\{A,\mathit{123}\}_{pkE})\cdot\mathrm{take}_{A\to E}(\{A,\mathit{123}\}_{pkE})\cdot\mathrm{fake}_{A\to B}(\{A,\mathit{123}\}_{pkB})\cdot\\
&\quad\mathrm{read}_{A\to B}(\{A,\mathit{123}\}_{pkB})\cdot\mathrm{send}_{B\to A}(\{\mathit{123},\mathit{456}\}_{pkA})\cdot\mathrm{take}_{B\to A}(\{\mathit{123},\mathit{456}\}_{pkA})\cdot\\
&\qquad\mathrm{fake}_{E\to A}(\{\mathit{123},\mathit{456}\}_{pkA})\cdot\mathrm{read}_{E\to A}(\{\mathit{123},\mathit{456}\}_{pkA})\cdot\mathrm{send}_{A\to E}(\{\mathit{456}\}_{pkE})\cdot\\
&\qquad\quad\mathrm{claim}_A(\mathit{ni\text{-}synch})\cdot\mathrm{take}_{A\to E}(\{\mathit{456}\}_{pkE})\cdot\mathrm{fake}_{A\to B}(\{\mathit{456}\}_{pkB})\cdot\\
&\qquad\qquad\mathrm{read}_{A\to B}(\{\mathit{456}\}_{pkB})\cdot\mathrm{claim}_B(\mathit{ni\text{-}synch}).
\end{aligned}$$

Note, for the particular choice of trace semantics here, the presence of take and fake actions in the former trace that model intruder activity. It is assumed that the intruder knows the private key of E, so that he can act as man-in-the-middle. Because A tries to communicate with the compromised agent E, we have that $\neg\,\mathit{Valid}(\mathrm{claim}_A)$. Of interest is the claim of B: we have that $\mathit{Valid}(\mathit{claim}_B)$.

A realistic choice for pe maps the trace to the sequence of protocol events

$$\begin{aligned}
&\mathit{send}_{I\to R}(\{I,nI\}_{pkR})\cdot\\
&\quad\mathit{read}_{I\to R}(\{I,nI\}_{pkR})\cdot\mathit{send}_{R\to I}(\{nI,nR\}_{pkI})\cdot\\
&\qquad\mathit{read}_{R\to I}(\{nI,nR\}_{pkI})\cdot\mathit{send}_{I\to R}(\{nR\}_{pkR})\cdot\\
&\qquad\quad\mathit{claim}_I(\mathit{ni\text{-}synch})\cdot\\
&\qquad\qquad\mathit{read}_{I\to R}(\{nR\}_{pkR})\cdot\mathit{claim}_R(\mathit{ni\text{-}synch}).
\end{aligned}$$

As the partial function pe is not defined on intruder trace events, they have vanished in the latter representation of the trace at the level of protocol events. In this trace, two role instances (runs) are involved. Thus, an equivalence relation π for $Tr(P)$ will distinguish the following subsets of trace events:

$$\begin{aligned}
&\{\,\mathrm{send}_{A\to E}(\{A,\mathit{123}\}_{pkE}),\ \mathrm{read}_{E\to A}(\{\mathit{123},\mathit{456}\}_{pkA}),\\
&\qquad\qquad\mathrm{send}_{A\to E}(\{\mathit{456}\}_{pkE}),\ \mathrm{claim}_A(\mathit{ni\text{-}synch})\,\}\\
&\{\,\mathrm{read}_{A\to B}(\{A,\mathit{123}\}_{pkB}),\ \mathrm{send}_{B\to A}(\{\mathit{123},\mathit{456}\}_{pkA}),\\
&\qquad\qquad\mathrm{read}_{A\to B}(\{\mathit{456}\}_{pkB}),\mathrm{claim}_B(\mathit{ni\text{-}synch})\,\}.
\end{aligned}\tag{4}$$

For a trace $t\in TE^*$, we write t_i for the i-th element of t. We use $<_t$ to denote the strict trace ordering of t, i.e., $t_i <_t t_j$ if $i<j$. The subscript is omitted from the notation $<_t$, if the trace t is clear from the context. Furthermore, for a trace event e, we write $e\in t$ in case $e=t_i$ for some index i.

The various notions of authentication discussed in this paper capture the interaction of the agents involved in the protocol. In general, the semantics allow for several protocol instances in a single trace in which several parties are involved. Typically, there are more parties than roles in a trace. Also, it can be very well the case that the same party is involved many times, possibly in different roles. Central is the question of who is doing what with whom. I.e., who are, from the perspective of a party involved in a particular protocol instance in a particular trace, the other parties involved in the protocol instance and in which roles?

The above is specifically relevant to the occurrences of claim trace events, as these events are intended to signify a moment in the trace where authentication has been established. Therefore, we introduce the notion of a cast, borrowing intuition from a theater play that is performed several times. The cast for a particular performance of the play relates activity in the performance to particular roles. Likewise, the concrete activities in a protocol instance at the trace level, are assigned to the roles in the protocol. In the theater case, in different performances an actor can play different roles. Also, the same role can be taken up, for different performances of the play, by different actors. Likewise, trace events associated with different roles may belong to the same agent and trace events that are instances for the same role may belong to different agents. For our purposes the coupling of roles and trace events is more important, than the coupling of roles and actors. Therefore, we have the following definition.

Definition 2 *[Cast] A mapping* $\Gamma\colon \mathit{ClaimTE} \to \mathit{Role} \to \mathcal{P}(\mathit{TE})$ *is called a* cast function *for the protocol* P *if*

$$e \in \Gamma(c)(\mathsf{R}) \implies \mathit{role}(\mathit{pe}(e)) = \mathsf{R} \wedge \Gamma(c)(\mathsf{R}) \subseteq [e]_\pi \tag{5}$$

for all $e \in \mathit{TE}$*,* $c \in \mathit{ClaimTE}$ *and* $\mathsf{R} \in \mathit{Role}$*, and*

$$c \in \Gamma(c)(\mathit{role}(c)). \tag{6}$$

for all $c \in \mathit{ClaimTE}$*. (Recall that* $\mathit{role}(c)$ *expands to* $\mathit{role}(\mathit{pe}(c))$ *when* c *is a trace event.)*

A cast function $\Gamma\colon \mathit{ClaimTE} \to \mathit{Role} \to \mathcal{P}(\mathit{TE})$ *is called an* injective *cast function, if*

$$\Gamma(c)(\mathsf{R}) = \Gamma(c')(\mathsf{R}') \implies c = c' \wedge \mathsf{R} = \mathsf{R}'$$

for every two claim trace events c*,* c' *and every two roles* R*,* R'*. We use* $\mathit{Cast}(P)$ *to denote the collection of all cast functions for the protocol* P*.*

The idea behind the notion of a cast function is the assignment of roles in the context of a concrete claim event c. The image $\Gamma(c)$ of Γ for c is a mapping from Role to $\mathcal{P}(\mathit{TE})$. The subset $\Gamma(c)(\mathsf{R})$ of TE consists of a number of trace events corresponding, in this cast, to the role R and falling within the same equivalence class of the partitioning π, as captured by condition (5). In particular, we require in condition (6) the claim event c to be one of the events attributed to the role of the claim. Note the inclusion $\Gamma(c)(\mathsf{R}) \subseteq [e]_\pi$ above. At this point, we do not require that every role will be performed completely, leaving room for unfinished role executions.

An injective cast function is, with abuse of language, an injective mapping when considered to be of functionality $\mathit{ClaimTE} \times \mathit{Role} \rightarrow \mathcal{P}(\mathit{TE})$, rather than injective as a function of type $\mathit{ClaimTE} \rightarrow \mathit{Role} \rightarrow \mathcal{P}(\mathit{TE})$. The main reason of sticking to the latter function type is the underlying intuition of a cast. The mapping $\Gamma(c)$ captures the perspective of the agent executing the instance of the role of the claim trace event c. From the point of view of the particular role of the agent, he expects certain activity to belong to particular roles, like, for the Needham-Schroeder case, "apparently agent A want to set up a session with me using nonce *123*" or "it must be the case that the initiator has received my response, for otherwise my nonce *456* would not have returned".

In the context of the Lowe attack trace introduced above, we have, e.g., as a cast the mapping Γ such that

$$\begin{aligned}
\Gamma(\mathrm{claim}_A(\mathit{ni\text{-}synch}))(I) = \{\ & \mathrm{send}_{A\rightarrow E}(\{A, \mathit{123}\}_{pkE}), \\
& \mathrm{read}_{E\rightarrow A}(\{\mathit{123}, \mathit{456}\}_{pkA}),\ \mathrm{send}_{A\rightarrow E}(\{\mathit{456}\}_{pkE}),\ \mathrm{claim}_A(\mathit{ni\text{-}synch})\ \} \\
\Gamma(\mathrm{claim}_A(\mathit{ni\text{-}synch}))(R) = \{\ & \mathrm{take}_{A\rightarrow E}(\{A, \mathit{123}\}_{pkE}), \\
& \mathrm{fake}_{E\rightarrow A}(\{\mathit{123}, \mathit{456}\}_{pkA}),\ \mathrm{take}_{A\rightarrow E}(\{\mathit{456}\}_{pkE})\ \} \\
\Gamma(\mathrm{claim}_B(\mathit{ni\text{-}synch}))(I) = \{\ & \mathrm{send}_{A\rightarrow E}(\{A, \mathit{123}\}_{pkE}), \\
& \mathrm{read}_{E\rightarrow A}(\{\mathit{123}, \mathit{456}\}_{pkA}),\ \mathrm{send}_{A\rightarrow E}(\{\mathit{456}\}_{pkE}),\ \mathrm{claim}_A(\mathit{ni\text{-}synch})\ \} \\
\Gamma(\mathrm{claim}_B(\mathit{ni\text{-}synch}))(R) = \{\ & \mathrm{read}_{A\rightarrow B}(\{A, \mathit{123}\}_{pkB}), \\
& \mathrm{send}_{B\rightarrow A}(\{\mathit{123}, \mathit{456}\}_{pkA}),\ \mathrm{read}_{A\rightarrow B}(\{\mathit{456}\}_{pkB}), \mathrm{claim}_B(\mathit{ni\text{-}synch})\ \}.
\end{aligned}$$

Note that, the cast Γ is bounded by the general restrictions of Definition 2 and the particular choice for the equivalence π (see formula (4)). For the claim trace event $\mathrm{claim}_A(\mathit{ni\text{-}synch})$ regarding the initiator role I, and the claim trace event $\mathrm{claim}_B(\mathit{ni\text{-}synch})$ regarding the responder role R, the images do not differ among different casts. For the trace under consideration, there are for $\Gamma(\mathrm{claim}_A(\mathit{ni\text{-}synch}))(R)$ and $\Gamma(\mathrm{claim}_B(\mathit{ni\text{-}synch}))(I)$, in principle alternatives. Because of the limited trace we consider, for $\Gamma(\mathrm{claim}_B(\mathit{ni\text{-}synch}))(I)$ this amounts to the same set of trace events. For $\Gamma(\mathrm{claim}_A(\mathit{ni\text{-}synch}))(R)$ the less plausible sequence comprising $\Gamma(\mathrm{claim}_B(\mathit{ni\text{-}synch}))(R)$ would do as well.

Regarding the first two clauses above, the completely genuine trace events of agent A are matched by activity of the intruder (who is assumed to have access to the private key of agent E). Just bad luck for agent A, but when takes and fakes are interpreted as reads and sends, nothing strange happens. Agent A, however, cannot distinguish between these two cases. The situation for the latter two clauses is more seriously wrong. Here, the role of responder of agent B, who assumes he is talking to agent A, is matched by activity of agent A engaged in an protocol session with agent E.

We now return to the main point of this paper: defining a strong form of authentication. We have established an abstract view on security protocol descriptions, and a second layer of trace events. We introduce an authentication property that captures the following correspondence: at the trace level, we require that the same structures occur as the ones found at the protocol description level. Informally put, we require that everything we intended to happen in the protocol description also actually happens in the trace.

We first give the definition of the authentication property that we call non-injective synchronisation, and explain it in detail below.

Definition 3 *[NI-SYNCH] A protocol P with a claim protocol event γ is called* non-injectively synchronising*, notation NI-SYNCH(P,γ), if*

$$
\begin{aligned}
&\forall t \in \mathit{Tr}(P)\ \exists \Gamma \in \mathit{Cast}(P)\ \forall c \in t,\ \mathit{Valid}(c),\ \mathit{pe}(c) = \gamma\\
&\quad \forall \sigma, \varrho\colon \sigma \rightsquigarrow \varrho \preccurlyeq_P \gamma\ \exists s, r\colon s <_t r <_t c\\
&\qquad \mathit{pe}(s) = \sigma\ \wedge\ s \in \Gamma(c)(\mathit{role}(\sigma)) \wedge\\
&\qquad \mathit{pe}(r) = \varrho\ \wedge\ r \in \Gamma(c)(\mathit{role}(\varrho)) \wedge\\
&\qquad \mathit{cont}(s) = \mathit{cont}(r).
\end{aligned}
$$

Non-injective synchronisation is a trace property for a protocol P and a claim role event γ. In particular, each trace t of the protocol P can contain a number of instances of the claim event γ. We only consider the valid instances of these claims, i.e. the claims of agents that communicate with agents that have not been compromised. For each of these instances of the claim, we require that there are actual communication partners. Thus, for all of these claims, there must exist runs that fulfill the roles of the protocol. This is expressed by the existence of the Γ function, which assigns the communication partner runs for each claim instance and role of the protocol.

Given an assignment of communication partners by the cast function Γ, we require, for each claim instance c, that the communications have occurred as expected. This requirement must hold for each communication pair (s,r) that precedes the claim role event, expressed as $\forall \sigma, \varrho : \sigma \rightsquigarrow \varrho \preccurlyeq_P \gamma$.

In order for the communication $\sigma \rightsquigarrow \varrho$ to have occurred correctly from the viewpoint of the claim trace event c, there must exist actual send and read events s and r, such that the following three conditions for correct communications hold: the order of the events must be correct, the events are instantiations of the right role events by the runs as defined in Γ, and the message must be communicated correctly.
For the first condition regarding the ordering in the trace, we require that just as in the protocol description, $s <_t r <_t c$ holds. The second condition requires that the trace events are indeed the events corresponding to the correct read and send events from the protocol, and that they are part of the runs as assigned by the Γ function. For the third condition, we simply require that the contents of the read message must be identical to the content of the sent message.

Concretely, for the case of the Needham-Schroeder protocol NS, we have that $\mathit{NI\text{-}SYNCH}$ does not hold for the claim event $\mathit{claim}_R(\mathit{ni\text{-}synch})$ of the responder. Consider again the trace

$$
\begin{aligned}
&\mathrm{send}_{A \to E}(\{A, \mathit{123}\}_{pkE}) \cdot \mathrm{take}_{A \to E}(\{A, \mathit{123}\}_{pkE}) \cdot \mathrm{fake}_{A \to B}(\{A, \mathit{123}\}_{pkB}) \cdot \\
&\quad \mathrm{read}_{A \to B}(\{A, \mathit{123}\}_{pkB}) \cdot \mathrm{send}_{B \to A}(\{\mathit{123}, \mathit{456}\}_{pkA}) \cdot \mathrm{take}_{B \to A}(\{\mathit{123}, \mathit{456}\}_{pkA}) \cdot \\
&\qquad \mathrm{fake}_{E \to A}(\{\mathit{123}, \mathit{456}\}_{pkA}) \cdot \mathrm{read}_{E \to A}(\{\mathit{123}, \mathit{456}\}_{pkA}) \cdot \mathrm{send}_{A \to E}(\{\mathit{456}\}_{pkE}) \cdot \\
&\qquad\quad \mathrm{claim}_A(\mathit{ni\text{-}synch}) \cdot \mathrm{take}_{A \to E}(\{\mathit{456}\}_{pkE}) \cdot \mathrm{fake}_{A \to B}(\{\mathit{456}\}_{pkB}) \cdot \\
&\qquad\qquad \mathrm{read}_{A \to B}(\{\mathit{456}\}_{pkB}) \cdot \mathrm{claim}_B(\mathit{ni\text{-}synch}).
\end{aligned}
$$

representing the Lowe attack considered before. We want to show that for this trace, there exists no cast for the claim trace event $\mathrm{claim}_B(\mathit{ni\text{-}synch})$ that satisfies $\mathit{NI\text{-}SYNCH}$. Given the clustering of trace events prescribed by the partitioning π and the requirement (6) of Definition 2, we have

$$
\begin{aligned}
\Gamma(\mathrm{claim}_B(\mathit{ni\text{-}synch}))(R) = \{\, &\mathrm{read}_{A \to B}(\{A, \mathit{123}\}_{pkB}), \\
&\mathrm{send}_{B \to A}(\{\mathit{123}, \mathit{456}\}_{pkA}),\ \mathrm{read}_{A \to B}(\{\mathit{456}\}_{pkB}), \mathrm{claim}_B(\mathit{ni\text{-}synch}) \,\}.
\end{aligned}
$$

We have as single possibility of $\Gamma(\mathrm{claim}_B(\mathit{ni\text{-}synch}))(I)$, by condition (5) of Definition 2,

$$
\begin{aligned}
\Gamma(\mathrm{claim}_B(\mathit{ni\text{-}synch}))(I) = \{\, &\mathrm{send}_{A \to E}(\{A, \mathit{123}\}_{pkE}), \\
&\mathrm{read}_{E \to A}(\{\mathit{123}, \mathit{456}\}_{pkA}),\ \mathrm{send}_{A \to E}(\{\mathit{456}\}_{pkE}),\ \mathrm{claim}_A(\mathit{ni\text{-}synch}) \,\}.
\end{aligned}
$$

However, none of the two read trace events will have counterparts in the run $\Gamma(\mathrm{claim}_B(\mathit{ni\text{-}synch}))(I)$ that match in content (assuming the public keys pkB

and pkE being different) as is required by the fact that

$$send_{I\to R}(\{I, nI\}_{pkR}) \rightsquigarrow read_{I\to R}(\{I, nI\}_{pkR}) \preccurlyeq_P claim_R(ni\text{-}synch) \wedge$$
$$send_{I\to R}(\{nR\}_{pkR}) \rightsquigarrow read_{I\to R}(\{nR\}_{pkR}) \preccurlyeq_P claim_R(ni\text{-}synch).$$

On the other hand, it is to be expected that $\mathit{NI\text{-}SYNCH}(NS, claim_I(ni\text{-}synch))$ holds. For the particular trace considered, the cast function Γ given as an instance of Definition 2 allows for a matching of the trace events corresponding to the protocol events preceding the claim $claim_I(ni\text{-}synch)$. However, for non-injective synchronisation to be satisfied, such a cast should exist for all traces in $Tr(NS)$, which we did not fully detail here.

Protocols that satisfy *NI-SYNCH* can still be vulnerable to so-called replay attacks, which will be explored in more detail in Section 4.1. Therefore, we modify *NI-SYNCH* slightly and require that the cast function is injective, yielding the notion of injective synchronisation.

Definition 4 *[I-SYNCH] A protocol P with a claim protocol event γ is called* injectively synchronising, *notation $\mathit{I\text{-}SYNCH}(P, \gamma)$, if*

$$\begin{array}{l} \forall t \in Tr(P)\ \exists \Gamma \in Cast(P),\ injective\ \forall c \in t,\ Valid(c),\ pe(c) = \gamma \\ \quad \forall \sigma, \varrho \colon \sigma \rightsquigarrow \varrho \preccurlyeq_P \gamma\ \exists s, r \colon s <_t r <_t c \\ \qquad pe(s) = \sigma\ \wedge\ s \in \Gamma(c)(role(\sigma)) \wedge \\ \qquad pe(r) = \varrho\ \wedge\ r \in \Gamma(c)(role(\varrho)) \wedge \\ \qquad cont(s) = cont(r). \end{array}$$

For the Needham-Schroeder protocol in Figure 1, *I-SYNCH* holds for the role I in the Dolev-Yao model: for each instance of the claim of role I in a trace, there must also be a unique instance of the role R to synchronise with.

3 Extending the authentication hierarchy

In [22], Lowe defined a number of authentication properties and positioned them in a hierarchy. In this section we study the relation between these properties and our notion of (injective) synchronisation. Since time is not considered in our model, we will restrict our attention to authentication properties not involving time.

The definitions provided by Lowe are all in an extensional style. For instance, agreement expresses that after successful completion of the protocol the parties agree on the values of all (or some) variables. In order to be able to compare this to our approach, a common framework is required. For this purpose, we tune the definitions of [22] here, to provide an intensional characterisation of agreement, to arrange for such a comparison. From these definitions it easily follows that injective synchronisation is stronger than injective agreement over all variables, and thus forms a new top element in the authentication hierarchy. Using these insights, we are able to show the difference between the several forms of authentication by means of some simple examples.

The starting point for providing an intensional characterisation of agreement is the following definition of *injective agreement* by Lowe [22].

> Initiator I is in agreement with responder R, whenever I as initiator completes a run of the protocol with R, then R as responder has been running the protocol with I. Moreover, I and R agree on all data variables, and each run of I corresponds to a unique run of R.

Although this definition is conceptually clear, it is still informal. Therefore, we will analyse this definition and translate the given concepts into our framework.

The main issue to be clarified is the notion of a *variable*, which is not defined in our approach. Since the values of the variables are determined by the contents of the messages sent and received, we can reformulate the correspondence between the variables as a requirement on the contents of the sent and received messages. In a two-party protocol, it is clearly the case that if two parties agree on the values of *all* variables, then they agree on the contents of all messages exchanged, and vice versa.

Agreement in a multi-party protocol means that only the initiator and the responder agree on their shared variables. There is no such requirement for the variables maintained by the other roles in the protocol. In order to be able to provide an intensional definition of agreement, we will have to extend the agreement relation to all parties involved in the protocol. Therefore, we will require that upon completion of the protocol *all* parties agree on *all* variables. This is somewhat stronger than the extensional definition provided by Lowe, but for many multi-party protocols this seems to be a natural extension. Summarising, we see that the agreement requirement translates to the demand that corresponding sends and receives have the same contents.

Given this interpretation of agreement, it is easy to see the correspondence with synchronisation. Like agreement, synchronisation requires correspondence on the contents of all messages, but it additionally requires that a message is sent before it can be received. The definition of Lowe does not bother about this send/read order. Thus, we arrive at the following definition of non-injective agreement, which is adapted from Definition 3 by removing the requirement that send events occur before their corresponding read event.

Definition 5 (*NI-AGREE*) *Given a protocol P with a claim protocol event γ, non-injective agreement holds, notation $\mathit{NI\text{-}AGREE}(P, \gamma)$, if*

$$\begin{aligned}
&\forall t \in \mathit{Tr}(P)\ \exists \Gamma \in \mathit{Cast}(P)\ \forall c \in t,\ \mathit{Valid}(c),\ \mathit{pe}(c) = \gamma \\
&\quad \forall \sigma, \varrho \colon \sigma \rightsquigarrow \varrho \preccurlyeq_P \gamma\ \exists s, r \colon s <_t c \wedge r <_t c \\
&\qquad \mathit{pe}(s) = \sigma\ \wedge\ s \in \Gamma(c)(\mathit{role}(\sigma)) \wedge \\
&\qquad \mathit{pe}(r) = \varrho\ \wedge\ r \in \Gamma(c)(\mathit{role}(\varrho)) \wedge \\
&\qquad \mathit{cont}(s) = \mathit{cont}(r)
\end{aligned}$$

The agreement predicate expresses that for all instantiated claims in any trace of a given security protocol, there exist runs for the other roles in the protocol, such that all communication events causally preceding the claim must have occurred before the claim.

Injective agreement is defined in the same way as injective synchronisation is obtained from non-injective synchronisation.

Definition 6 (*I-AGREE*) *Given a protocol P with a claim protocol event γ, injective agreement holds, notation $\mathit{I\text{-}AGREE}(P, \gamma)$, if*

$$\begin{aligned}
&\forall t \in \mathit{Tr}(P)\ \exists \Gamma \in \mathit{Cast}(P),\ \mathit{injective}\ \forall c \in t,\ \mathit{Valid}(c),\ \mathit{pe}(c) = \gamma \\
&\quad \forall \sigma, \varrho \colon \sigma \rightsquigarrow \varrho \preccurlyeq_P \gamma\ \exists s, r \colon s <_t c \wedge r <_t c \\
&\qquad \mathit{pe}(s) = \sigma\ \wedge\ s \in \Gamma(c)(\mathit{role}(\sigma)) \wedge \\
&\qquad \mathit{pe}(r) = \varrho\ \wedge\ r \in \Gamma(c)(\mathit{role}(\varrho)) \wedge \\
&\qquad \mathit{cont}(s) = \mathit{cont}(r)
\end{aligned}$$

It expresses that for any trace and for any run of any role in the protocol there exist unique runs for the other roles of the protocol such that for all claims occurring in the trace all communications preceding the claim must have occurred correctly within these runs.

The definition of *I-AGREE* does not involve *all* communications, but only the set of events that causally precede a claim. However, it turns out that the way in which agreement is made precise in terms of CSP, as can be checked by compiling Casper-code into CSP, it also takes only preceding communications into account. For this, running-commit signals (see [28]) are introduced in the protocol. For each role, a running signal is added to the last communication preceding the agreement claim. In the role that includes the claim, a commit signal is added to the last communication. Injective agreement over all roles requires that the running signals of each role precede the commit signal. This corresponds to the order requirements of *I-AGREE*.

The definitions of the four security properties above clearly reveal their relative strengths in excluding attacks. Every injective protocol is also non-injective and if a protocol satisfies synchronisation then it satisfies agreement too. Figure 2 shows this hierarchy. An arrow from property X to property Y means that every protocol satisfying X also satisfies Y. Phrased differently, the class of protocols satisfying X is included in the class satisfying Y.

$$\begin{tikzcd}
& \mathit{I\text{-}SYNCH} \arrow[dl] \arrow[dr] & \\
\mathit{NI\text{-}SYNCH} \arrow[dr] & & \mathit{I\text{-}AGREE} \arrow[dl] \\
& \mathit{NI\text{-}AGREE} &
\end{tikzcd}$$

Fig. 2. Hierarchy of security properties.

The correctness of the hierarchy is captured by the following theorem.

Theorem 1 *The security properties I-SYNCH, NI-SYNCH, I-AGREE, and NI-AGREE satisfy the inclusion relation as depicted in Figure 2.*

Proof. Straightforward from the definitions. □

The question of whether the inclusions in Figure 2 are strict is harder to answer. In part, this is due to the abstractness of our model. Since our approach is parameterised over the actual semantics, and thus over the intruder model, we cannot determine for a given protocol to which class it belongs. Therefore, strictness of the inclusions can only be answered relative to a given semantics. Consequently, the following reasoning will be at a conceptual level only.

If we take, e.g., a model where all agents simply follow their roles and the intruder has no capabilities at all, then the diamond in Figure 2 collapses into a single class. The same holds if the intruder can only eavesdrop on the communications. However, in the Dolev-Yao model, all inclusions are strict, as we will see below.

The case of injectivity vs. non-injectivity has been studied extensively before. The MSC on the left in Figure 3 shows a protocol that satisfies *NI-SYNCH* and *NI-AGREE*, but neither *I-SYNCH*, nor *I-AGREE*.

The intruder will only be able to construct message $\{I, R\}_{skI}$ after having

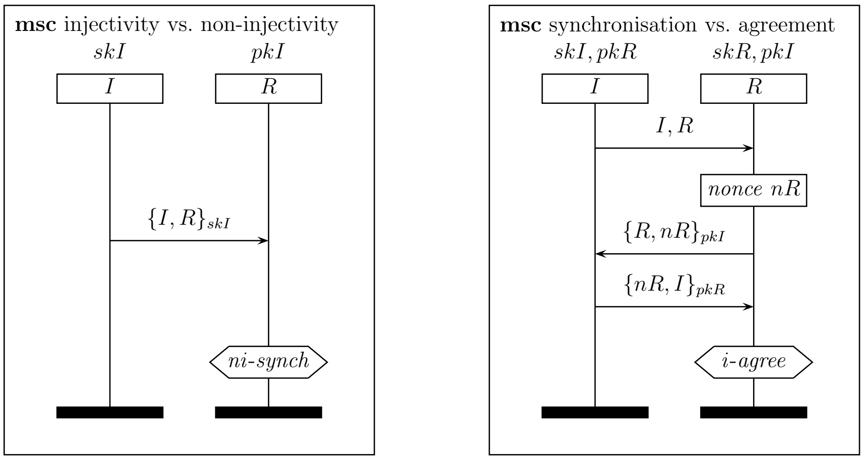

Fig. 3. Distinguishing protocols

eavesdropped this message from a previous run. Therefore every read event of this message is preceded by a corresponding send event, so the protocol is both *NI-SYNCH* and *NI-AGREE*. However, once the intruder has learnt this message, he can replay it as often as desired, so the protocol is not injective.

A distinguishing example between synchronisation and agreement is depicted on the right in Figure 3. As confirmed by the Casper/FDR tool set, this protocol satisfies unilateral authentication in the sense of agreement (both injective and non-injective). However, the protocol does not satisfy synchronisation (both variants). This is the case, because the intruder can send message I, R long before I actually initiates the protocol, making R to believe that I has requested the start of a session before he actually did.

The two examples show that the inclusions of the diamond in Figure 2 are strict if the intruder has the capabilities to eavesdrop, deflect and inject messages. Both examples also imply that there are no arrows between *NI-SYNCH* and *I-AGREE*.

3.2 Synchronisation vs. agreement

As stated before, the difference between synchronisation and agreement is rather subtle and, indeed, most authentication protocols in practice satisfy both properties. The distinction is that synchronisation requires that corresponding send and receive messages have to be executed in the expected order, while for agreement a message may be received before it is sent. This can, for instance, be caused by a message injection of the intruder. An attack in which the intruder injects a message before its actual creation is called a *preplay* attack. Whether such a protocol weakness can be exploited by the attacker depends on the intention of the protocol. Below we will sketch three examples of possible weaknesses.

In the first example we consider the notion of *predictable nonces*. There may be several reasons for such predictability, such as a bad pseudo-random generator, or the fact that the nonce is implemented as a counter. Consider, for instance, the protocol in Figure 4. The purpose of this protocol is unilateral authentication of responder R towards initiator I. This is established by using nonce nI as a challenge. However, if the value of this nonce is predictable by the intruder, the protocol has a major shortcoming. This is shown in the trace in the right of Figure 4, which displays that the run of R has finished even before the run of I started. The consequences of this preplay attack are similar to many well-known replay attacks. The protocol satisfies (injective and non-injective) agreement, but does not satisfy synchronisation (both variants).

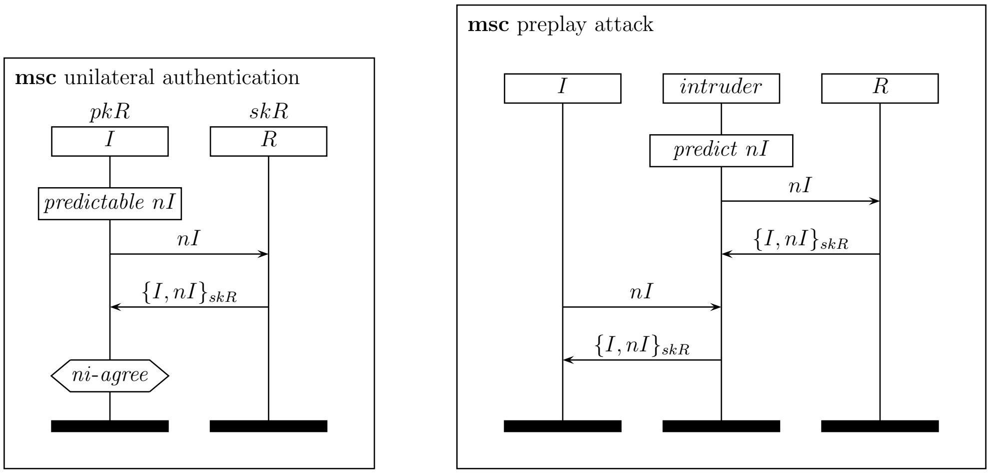

Fig. 4. Preplay attack due to a predictable nonce.

This type of preplay attacks is also called *suppress-replay* attacks [14,24]. As pointed out by Chen and Mitchell [24], practical protocols such as the S/KEY user authentication scheme suffer from this kind of attack because they use predictable challenges. Roscoe [27] found a similar problem for the Needham-Schroeder Secret Key protocol in the case that the initiator's nonce is predictable.

The second example concerns the protocol in Figure 5, in which we assume that an agent keeps a state which is shared by all its instances of the protocol. The purpose of this protocol is again unilateral authentication, but now the responder is in control of the nonce. After receiving a request from the initiator, the responder sends his nonce to the initiator. The initiator keeps a set V in which he stores all nonces from previous instantiations of the protocol. This is to prevent replay attacks and, thus, to ensure injectivity. If the nonce is accepted as fresh, the initiator challenges the responder to prove his identity, which the responder does by replying the signed nonce. This may seem a reasonable authentication protocol, and indeed it satisfies injective agreement. However, the preplay attack shown in Figure 5 indicates a weakness of the protocol. An initiator can successfully execute his side of the protocol, while the responder was not even alive when the initiator started the protocol. This example shows that even a complex message interaction can be preplayed. Since the protocol does not satisfy synchronisation, this weakness can be detected by verifying synchronisation.

Two remarks apply to this example. The first remark is that testing whether the nonce is already in V and the extension of V with the nonce should be implemented as an atomic action (a *test-and-set* action) to achieve the desired result. The second remark is that this protocol still has interesting properties when leaving out the validation of nR's freshness by the initiator (i.e. if we remove the set V and its operations). The resulting protocol is not injective anymore, since the intruder may replay the responder behaviour of an earlier run of the protocol. However, this reduced protocol still satisfies non-injective agreement. Since the displayed attack remains possible, the protocol does not satisfy synchronisation. Thus, we have a stateless protocol suffering from a preplay attack. It satisfies non-injective agreement but it does not satisfy non-injective synchronisation.

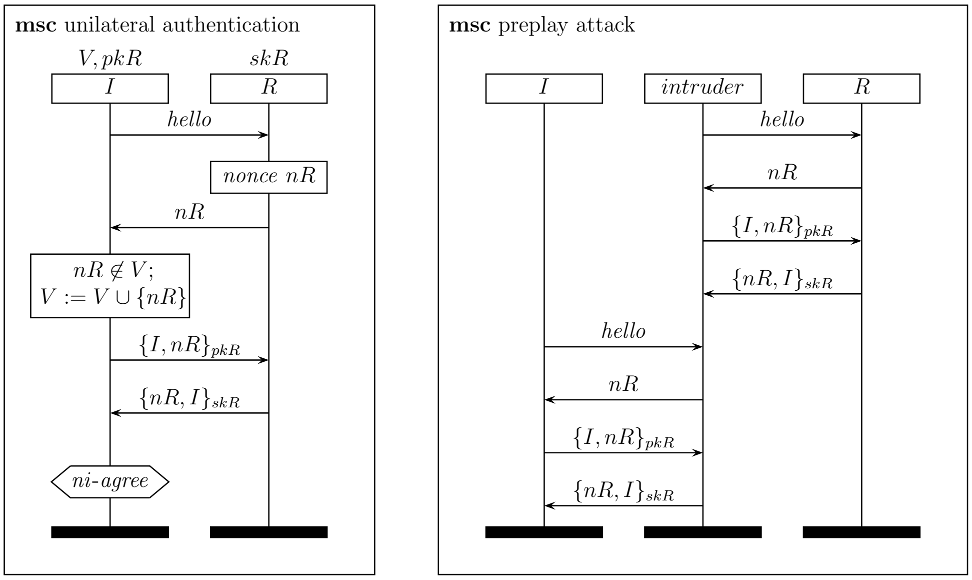

Fig. 5. Preplay attack because the nonce is controlled by the responder.

In the previous two examples we have seen how the intruder can preplay a complete protocol session and use it later to fool the initiator into thinking that the responder is still alive. In the third example, we see that it can already be harmful if only a single message is preplayed by the intruder.

In Figure 6 R is an Internet Service Provider, used by I. Assume that I pays R for the time he is connected. When I wants to connect, R retrieves the certificate of I from the trusted server S and uses this to authenticate I. After a successful session, I is billed from the moment the first message was received by R.

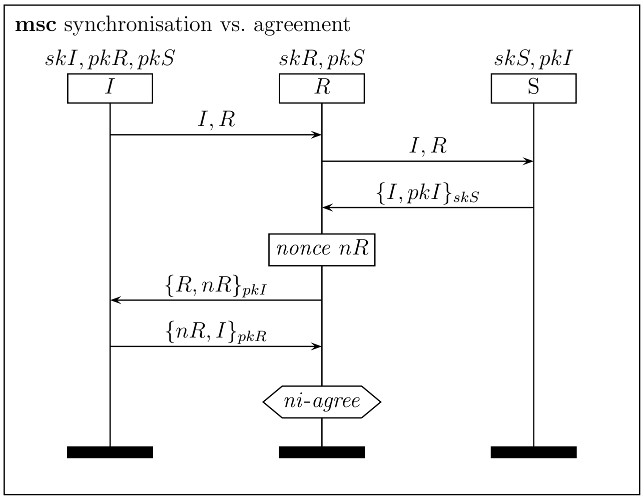

Fig. 6. A protocol satisfying *NI-AGREE* but not *NI-SYNCH*.

This protocol is a slightly modified version of the Needham-Schroeder-Lowe protocol. It can be exploited as follows. An intruder can send the first message pre-emptively, causing R to initiate a session with what it believes is I. If at some later time I decides to initiate a session with R and finishes it successfully, I will receive a bill that is too high. In fact, although, this protocol satisfies agreement for R, the first message is not authenticated at all. In contrast, this protocol does not satisfy synchronisation. The protocol can be easily modified to satisfy *NI-SYNCH* and thus to be resilient against the sketched type of timing attacks.

This type of attack may seem of little relevance, but it depends on the interpretation of the messages whether such unexpected behaviour can cause harm or not. In the rather contrived example above, a typical interpretation of the messages concerning billing time allows the intruder to exploit this unexpected behaviour. Following the observations of Roscoe [27], this sort of behaviour, while seemingly innocent was certainly unexpected by the protocol designer. After finding such unexpected behaviour, the designer has two options. First, he may decide that this behaviour is acceptable, but then he has to take the implications of this behaviour into account and extend his mental model of the protocol. He should make sure that this behaviour does not interfere with any other analysis which is based on the intended order of the protocol events. Still according to Roscoe, the second option is to strengthen the protocol as to make it compatible with the mental model again. As pointed out by Roscoe, the TMN protocol, and a seemingly correct strengthening thereof, suffer from the same weakness as the protocol in Figure 6.

We conclude by stating that failure of a protocol to respect synchronisation does not always indicate an exploitable weakness of the protocol. However, such unexpected behaviour should always receive extra attention and should at least lead to adjusting the mental model of the protocol.

4 Verifying injective synchronisation

Several tools exist to verify whether a protocol satisfies some form of agreement, e.g. [21,3,29]. Because synchronisation is very similar to agreement, we expect that it will be feasible to adapt most of the verification tools to be able to handle at least non-injective synchronisation.

In refinement or forward model-checking approaches, agreement is commonly verified by inserting *running* and *commit* or similar signals in the protocol. When somebody commits to some values, the other party needs to have emitted a running signal. The commit signal corresponds to the claim in our framework, whereas the running signal denotes the last communication of the other role that causally precedes the claim. These signals are introduced to ease verification: instead of having to inspect the trace leading up to the claim, only the set of emitted signals needs to be inspected. In our framework, agreement is a property of the trace prefix ending in a claim. By introducing running and commit signals, agreement can be verified by inspecting the set of signals. In much the same way, it is possible to verify synchronisation by introducing such signals for each communication that precedes the claim.

We have developed a protocol verification tool that can verify non-injective agreement and non-injective synchronisation as defined here (see [6]). The tool can compute trace prefixes of a protocol leading up to a claim. Given such a trace prefix, verification involves checking whether a suitable Γ exists such that the communications have occurred as expected.

In order to verify injective agreement, many tools rely on a counting argument: if a trace prefix contains n commit signals, there should be at least n preceding running signals in the trace prefix. If enough commit signals are considered in this way, this should ensure injectivity. The main drawback of this method is that the verification complexity is exponential in the length of the traces, and thus in the number of running claims considered. Another approach to verifying injectivity uses detailed knowledge of the data model: if the agreement includes data items that are guaranteed to be unique for each instance of the claim, injectivity can be derived. However, not all injective protocols include such unique data items. In some cases it can also be non-trivial to establish the required uniqueness property, as we show in the next section.

This section will focus on the verification of injectivity for protocols that satisfy non-injective synchronisation. We propose and study a property, the *LOOP* property, that can be syntactically verified. We prove a theorem that shows that *LOOP* is sufficient to guarantee injectivity. Our result is generic in the sense that it holds for a wide range of security protocol models, and does not depend on the details of message content or nonce freshness.

The remainder of this section will proceed as follows. In Section 4.1 we elaborate on the difficulties posed by injectivity, and informally describe our theorem. In Section 4.2 we define a class of security protocol models for which our theorem holds. Then, in Section 4.3 we propose and study the *LOOP* property, and prove the main theorem.

4.1 Injectivity of synchronisation

As stated in Section 1, protocols satisfying non-injective synchronisation may still be vulnerable to so-called *replay attacks*. In a replay attack the intruder replays a message taken from a different context, thereby fooling the honest participants into thinking they have successfully completed the protocol run. See [23].

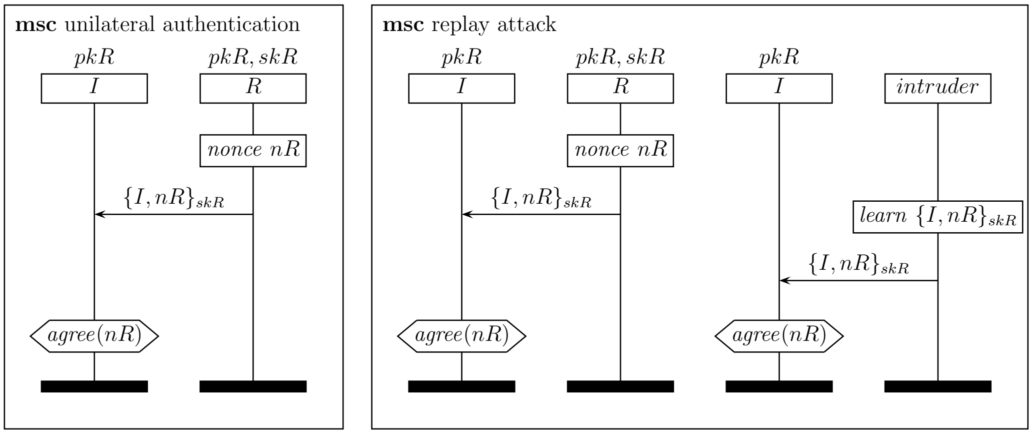

Fig. 7. An authentication protocol that is vulnerable to a replay attack.

The left-hand protocol in Figure 7 shows an example of a protocol where the parties agree upon the values of the variables (i.e. nonce nR), while the right-hand scenario shows a replay attack on this protocol. The intruder can overhear the message sent and can fool I in a future run to think that R has sent this message.

In order to rule out such flawed protocols, the additional property of *injectivity* is required. The unilateral authentication protocol from Figure 7 clearly does not satisfy injectivity, as is shown by the replay attack in the right-hand side of the figure. A simple fix would be to have the initiator determine the value of the nonce, as in Figure 8.

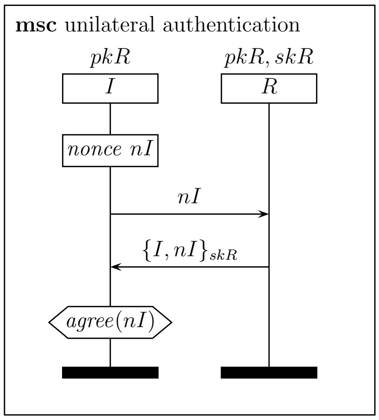

Fig. 8. Fixing the injectivity problem.

The introduction of a causal chain of messages from the initiator to the responder and back to the initiator seems to do the trick. We will call such a chain a *loop*. The presence of such a loop plays a key role in the discussion on injectivity below.

It is folklore that a nonce handshake is sufficient to ensure injectivity. Here we identify a more abstract property, viz. the occurrence of a loop, which is independent of the data model, and thus applicable to a wide range of security protocol models. To give an indication of the limitations of the data based approach, consider a protocol where a nonce n is created, and some function is applied to it. The result $f(n)$ is sent to the responder, who applies another function and replies with $g(f(n))$. Now, to check whether such a protocol can be injective based on the freshness of n in a data-based model, we need to know some details of f and g. If for example $f(x) = x \mod 2$, the protocol will not be injective. Therefore, we propose to only reason about loops, which does not require any information about the contents of messages.

In the next subsection we study the question whether there is a method to validate injectivity of a security protocol which is generic in the sense that it can be applied within a large class of verification methodologies, regardless of the data model that is used. Our main result is that, for a large class of security protocol semantics, the *LOOP* property introduced above guarantees that a synchronising protocol is also injective.

4.2 A class of security protocol models

The class of security protocol semantics for which our result holds, is characterised by the closure of the set of execution traces under swapping of events. This class contains, e.g., the process algebraic approach with the standard Dolev-Yao intruder model. Apart from this swap property, we will need no other assumptions on the data model and the intruder model. Since the *LOOP* property can easily be verified by means of static analysis of the security protocol description, we provide, in fact, a practical syntactic criterion for verifying injectivity.

We introduce the notation $t = u1; u2$ to denote that t is a concatenation of trace event sequences $u1$ and $u2$.

Definition 7 (Swap) *A security protocol semantics satisfies the SWAP property if the following two conditions hold:*

(i) The trace set $Tr(P)$ is closed with respect to non-read swaps, i.e., for all trace events e' such that $pe(e') \notin ReadPE$ it holds that

$$t; e; e'; t' \in Tr(P) \ \wedge \ e \not\sim_\pi e' \ \Rightarrow \ t; e'; e; t' \in Tr(P)$$

for all trace events e and traces t, t'.

(ii) The trace set $Tr(P)$ is closed with respect to read swaps, i.e., for s, r, e with $pe(s) \in SendPE$, $pe(r) \in ReadPE$ we have that

$$t; s; t'; e; r; t'' \in Tr(P) \ \wedge \ cont(s) = cont(r) \ \wedge \ e \not\sim_\pi r \\ \Rightarrow \ t; s; t'; r; e; t'' \in Tr(P)$$

for all traces t, t', t''.

These properties state that we can shift a non-read event to the left as long as it does not cross any other events of the same role instance. For the read event we have an additional constraint: we can only shift it to the left if there remains an earlier send of the same message.

For the remainder of this section we assume the protocol P contains a claim γ. We introduce a predicate χ and a set χ' for protocols that satisfy non-injective agreement for this claim γ. Given a trace t, a claim event c, and a cast Γ that maps the roles to runs, we express the auxiliary predicate χ on the domain

$Tr(P) \times Cast(P) \times ClaimTE$ by

$$\begin{aligned}
\chi(t,\Gamma,c) \iff & \ pe(c) = \gamma \wedge \\
& \forall \sigma, \varrho : \sigma \rightsquigarrow \varrho \wedge \varrho \preccurlyeq_P \gamma \ \exists s, r : s <_t r <_t c \\
& \qquad pe(s) = \sigma \ \wedge \ s \in \Gamma(c)(role(\sigma)) \wedge \\
& \qquad pe(r) = \varrho \ \wedge \ r \in \Gamma(c)(role(\varrho)) \wedge \\
& \qquad cont(s) = cont(r)
\end{aligned}$$

The first conjunct of this predicate expresses the fact that the run executing the claim role is determined by the parameter c. The second conjunct expresses that in the trace t, the claim c is valid with respect to the specific cast Γ, i.e., that the partners have executed all communications as expected. In the formula this is expressed by the fact that send and read events are executed by the expected runs, viz. $\Gamma(c)(role(pe(s)))$ and $\Gamma(c)(role(pe(r)))$, respectively, with identical contents, and in the right order.

Given a valid synchronisation claim c in a trace t, there exists a role instantiation function Γ such that $\chi(t,\Gamma,c)$ holds. The predicate χ tells us that certain events exist in the trace. Because we want to reason about these events in the following, we decide to make this set of events explicit. We define the set of events $\chi'(t,\Gamma,c)$ by

$$\chi'(t,\Gamma,c) = \{ \ e \in t \mid e \in \Gamma(c)(role(e)) \ \wedge \ pe(e) \preccurlyeq_P pe(c) \ \}$$

Assuming that χ holds, its set of events χ' has two interesting properties. If there is a read in this set, there is also a matching send in the set. Furthermore, given an event of a role in the set, all preceding events of the same role are also in the set.

To prove our main result, the $SWAP$ property from Definition 7 suffices. However, to ease the explanation of the proof, we introduce two additional lemmas. These lemmas are implied by the model and the two swap conditions.

The first lemma generalises the swapping of two events to the swapping of a set of events. The lemma does not hold for any set of events: we now use results obtained for a set of events defined by χ, that are involved in a synchronisation claim. Based on the two swap properties, we can shift these events (in their original order) to the beginning of the trace. This trace transformation function $shift \colon \mathcal{P}(TE) \times TE^* \to TE^*$ is defined by

$$shift(E,t) = \begin{cases} t & \text{if } t \cap E = \emptyset \\ e; shift(E, u1; u2) & \text{if } t = u1; e; u2 \wedge u1 \cap E = \emptyset \wedge e \in E \end{cases}$$

Here, the intersection of a trace and a set yields the collection of elements of the set occurring in the trace. This function effectively reorders a trace. The next lemma formulates conditions assuring that the reordering of a trace in $Tr(P)$ is in $Tr(P)$ as well.

Lemma 1 *Given a protocol P and a trace $t \in Tr(P)$, claim event c and role instantiation function Γ:*

$$\chi(t, \Gamma, c) \ \wedge \ t' = \mathit{shift}(\chi'(t, \Gamma, c), t) \ \Rightarrow t' \in Tr(P) \ \wedge \ \chi(t', \Gamma, c).$$

Proof. Induction on the size of the finite set $\chi'(t, \Gamma, c)$, because $\chi(t, \Gamma, c)$ implies that the read events can be swapped. Recall that, by convention, each event occurs at most once in a trace. □

The lemma directly generalises to more claim instances (of the same claim). Thus, instead of a single claim run, we can consider sets of claim runs.

Lemma 2 *Given a trace t, a set of claim events $C \subseteq t$ and cast $\Gamma \in Cast(P)$:*

$$(\forall c \in C \colon \chi(t, \Gamma, c)) \ \wedge \ t' = \mathit{shift}(\bigcup_{c \in C} \chi'(t, \Gamma, c), t) \ \Rightarrow$$
$$t' \in Tr(P) \ \wedge \ (\forall c \in C \colon \chi(t', \Gamma, c))$$

Proof. Similar to the proof of Lemma 1. □

If we apply the *shift* function to a trace of the system, and the conditions of the lemma are met, we get a reordered trace, that is also in $Tr(P)$. The new trace consists of two segments: in the first segment there are only the preceding events of the claim events in C, and all other events are in the second segment.

Intuitively, these lemmas express that the events involved in a valid synchronisation claim are independent of the other events in the trace. A valid synchronisation can occur at any point in the trace, because it does not require the involvement of other runs, or of the intruder. However, other events in the trace might depend on events involved in the synchronisation. Although we cannot shift the synchronising events to the right, we can shift them to the left, which ensures that any dependencies will not be broken.

We use Lemma 1 and Lemma 2 in the injectivity proof in the next section.

We define a property of protocols, which we call the *LOOP* property. For protocols with only two roles, it resembles a ping-pong property. First the claim role executes an event, then the other role, and then the claim role again. For example, the *LOOP* property does not hold for the protocol in Figure 7, but it does hold for the protocols in Figures 8 and 1.

We generalise this for multi-party protocols with any number of roles. We require that the partner roles have an event that must occur after the start of the claim run, but before the claim event itself.

Definition 8 *A security protocol P has the LOOP property with respect to a claim γ if*

$$\forall \varepsilon \preccurlyeq_P \gamma, \mathit{role}(\varepsilon) \neq \mathit{role}(\gamma) \\ \exists \varepsilon', \varepsilon'' \colon \varepsilon' \preccurlyeq_P \varepsilon'' \preccurlyeq_P \gamma \ \wedge \ \mathit{role}(\varepsilon') = \mathit{role}(\gamma) \ \wedge \ \mathit{role}(\varepsilon'') = \mathit{role}(\varepsilon). \quad (7)$$

The property tells us that for each role that has an event ε that precedes the claim γ, there exists a loop from the claim role to the role and back. This structure is identified in the formula by $\varepsilon' \preccurlyeq_P \varepsilon'' \preccurlyeq_P \gamma$.

Lemma 3 *Given a security protocol P with a claim γ: If all roles $\mathsf{R} \neq \mathit{role}(\gamma)$ that have events preceding γ, start with a read event, then we have that $\mathit{LOOP}(P, \gamma)$.*

The proof of this lemma follows from the definition of the protocol order $\preccurlyeq_P$: if a role R has an event ε'' that precedes the claim γ, and starts with a read event, then there must be a preceding event on some other role. Because all roles except for the claiming role start with a read, and role definitions are non-cyclic and finite, there must ultimately exist an event ε' of role $\mathit{role}(\gamma)$ that precedes ε'', and thus we can conclude that $\mathit{LOOP}(P, \gamma)$ holds.

In practice, this lemma tells us that the *LOOP* property always holds for the initiating role of a protocol. Thus, we only have to check whether the *LOOP* property holds for responder roles.

Now we can state a theorem, which provides a syntactic condition for the injectivity of a protocol that synchronises.

Theorem 2

$$\mathit{NI\text{-}SYNCH}(P, \gamma) \wedge \mathit{LOOP}(P, \gamma) \Rightarrow \mathit{I\text{-}SYNCH}(P, \gamma)$$

Proof. By contradiction. Assume that the implication does not hold. Thus we have

$$\textit{NI-SYNCH}(P,\gamma) \wedge \textit{LOOP}(P,\gamma) \wedge \neg \textit{I-SYNCH}(P,\gamma). \tag{8}$$

The remainder of the proof is done in two steps. The first step of the proof establishes a trace t of the protocol, in which there are two runs that synchronise with the same run. In the second step we use the shifting lemmas to transform t into another trace of the protocol. For this new trace, we will show that *NI-SYNCH* cannot hold, which contradicts the assumptions.

From now on, we will omit the type information for t and Γ in the quantifiers and assume that $t \in \mathit{Tr}(P)$.

Given that the protocol synchronises, but is not injective, we derive from definition 3 and 4 and formula (8) that

$$\begin{aligned} &\forall t\ \exists \Gamma\ \forall c \in t : \mathit{pe}(c) = \gamma \Rightarrow \chi(t,\Gamma,c) \wedge \\ &\qquad \neg \forall t\ \exists \Gamma \text{ injective } \forall c \in t : \mathit{pe}(c) = \gamma \Rightarrow \chi(t,\Gamma,c) \end{aligned} \tag{9}$$

We push the negation on the right through the quantifiers, yielding

$$\begin{aligned} &\forall t\ \exists \Gamma\ \forall c \in t : \mathit{pe}(c) = \gamma \Rightarrow \chi(t,\Gamma,c) \wedge \\ &\qquad \exists t\ \forall \Gamma \neg(\Gamma \text{ injective} \wedge\ \forall c \in t : \mathit{pe}(c) = \gamma \Rightarrow \chi(t,\Gamma,c)). \end{aligned} \tag{10}$$

Based on the existential quantifiers in (10), we choose a trace t and instantiation function Γ such that

$$\begin{aligned} &\forall c \in t : \mathit{pe}(c) = \gamma \Rightarrow \chi(t,\Gamma,c) \wedge \\ &\qquad \neg(\Gamma \text{ injective} \wedge\ \forall c \in t : \mathit{pe}(c) = \gamma \Rightarrow \chi(t,\Gamma,c)). \end{aligned} \tag{11}$$

Note that in (11) the left conjunct also occurs as a sub-formula in the right conjunct. Rewriting yields

$$\forall c \in t : \mathit{pe}(c) = \gamma \Rightarrow \chi(t,\Gamma,c)\ \wedge\ \neg(\Gamma \text{ injective}). \tag{12}$$

Making the non-injectivity for the function Γ explicit as explained in Definition 2, there must exist two claim events, for which χ holds:

$$\begin{aligned} &\exists c1, c2, R1, R2 : \chi(t,\Gamma,c1)\ \wedge\ \chi(t,\Gamma,c2) \\ &\qquad \wedge\ \Gamma(c1)(R1) = \Gamma(c2)(R2)\ \wedge\ (c1 \neq c2\ \vee\ R1 \neq R2) \end{aligned} \tag{13}$$

From the predicate χ and formula (13), we have that the run $\Gamma(c1)(R1)$ must be executing the role $R1$. Because $\Gamma(c1)(R1) = \Gamma(c2)(R2)$ it is also executing role $R2$. As defined in Section 2, runs only execute a single role, and we derive that $R1 = R2$. The fourth conjunct now reduces to $c1 \neq c2$.

Put $R = R1 = R2$. We choose two claim events $c1, c2$ such that Formula (13) holds for R. Now there exists a trace event set S such that

$$\Gamma(c1)(R) = \Gamma(c2)(R) = S$$

From the definition of χ, we obtain that if R would be equal to $role(\gamma)$, we would have $S = c1$ and $S = c2$, implying $c1 = c2$ and contradicting Equation (13). Thus, we have $R \neq role(\gamma)$.

We have now established that the trace t contains events from at least three role instances. Two of these, $[c1]_\pi$ and $[c2]_\pi$, are executing the claim role, while the third, S is executing a different role R. Furthermore, we have that the claims $c1$ and $c2$ synchronise with S.

This completes the first step of the proof. We will now proceed by transforming t into a trace for which *NI-SYNCH* cannot hold, for the second part of the proof.

Because we have $\chi(t, \Gamma, c1)$ and $\chi(t, \Gamma, c2)$, on the basis of Lemma 2 we can apply *shift* using $c1$ and $c2$ to get a trace $t' \in Tr(P)$

$$t' = shift(\chi'(t, \Gamma, c1) \cup \chi'(t, \Gamma, c2), t)$$

In the trace t' we now have two distinct segments. All events involved with the synchronisation of $c1$ and $c2$ are now in the initial segment of t'. This includes the events of S that precede the claim. The second segment of t' contains all other events, that are not involved in the preceding events of $c1$ and $c2$.

We will now reorder the initial segment of t'. To this end, we apply the *shift* function a second time, now only for $c1$. This will also yield a trace of the protocol, because the conditions of Lemma 2 hold for t', as the application of *shift* to t maintained the order of the events in the shifted set, which implies that $\chi(t', \Gamma, c1)$ holds. Thus, we also know that the trace t'' is an element of $Tr(P)$, where

$$t'' = shift(\chi'(t', \Gamma, c1), t')$$

Because the *shift* function maintains the order of the involved events, we have that $t'' = u1; u2; u3$, where

$$\begin{aligned} set(u1) =& \chi'(t', \Gamma, c1) \\ set(u2) =& \chi'(t, \Gamma, c2) \setminus \chi'(t', \Gamma, c1) \end{aligned}$$

All events that are not involved with the synchronisation claims $c1$ and $c2$, are in $u3$.

Observe that $u1$ includes all events of S that are involved with the claim of the run $c1$. As all events are unique, these are not part of $u2$. From the construction of the involved events set, we know that all involved events of role R are also in S, because all other role instances are executing other roles (as indicated by Γ). This implies that there are no events of role R in the $u2$ segment at all: these are all in $u1$.

Now we have arrived at a contradiction. t'' is in the set $Tr(P)$. The loop property combined with *NI-SYNCH* requires that for each role, there is an event after the first event of the claim role, that occurs before the claim. For the run $c2$ all events are in $u2$ (including the start and the claim), but in this segment there is no event of role R. Thus, there can be no Γ for t'' such that $\chi(t'', \Gamma, c2)$ holds. This implies that *NI-SYNCH* does not hold for the protocol, which contradicts the assumptions. □

Thus, we have established that *LOOP* is a sufficient condition to guarantee injectivity for protocols that satisfy *NI-SYNCH*.

5 Related work

Historically, many different interpretations of authentication exist. Here we focus on authentication of communication protocols. Early authentication concepts simply mention that one "end" of the channel can assure itself regarding the identity of the other end, as in e.g. [25,26]. An identity is considered to be an "end". These concepts seem very similar to Lowe's definition of *aliveness* in [22]. For this form of authentication, it is only required that the party to be authenticated performs some action to prove his identity (i.e. applying his secret key) regardless of the context, or whom he is proving his identity to. This is a rather weak form of authentication.

Some more advanced notions of authentication can be found in [19], for example, where mechanisms are sketched to prevent *message stream modification*. This can be prevented by achieving three subgoals: determine message authenticity, integrity, and ordering. Although no formalisation is provided, this concept is akin to our notion of non-injective synchronisation.

The international standard ISO/IEC 9798-1 [11] states that authentication requires verification of an entity's claimed identity. In response, Gollmann points out in [13] that the concept of a sender of a message should be treated with caution, and that replays should be prevented. Gollmann argues that authentication of an entire communication session is done by first setting up a session key, and that further messages are authenticated on the basis of this key. Based on this assumption, four authentication goals are identified. These goals explicitly assume that the protocols are implemented using private keys and session keys. Our definition of injective synchronisation is independent of protocol details, though. Therefore, it can be applied to a wider range of protocols.

An alternative formulation of authentication is given by Diffie, Oorschot, and Wiener in [8]. Here, participants are required to have matching message histories. The message history is allowed to be partial, if the last message sent is never received, which corresponds to our notion of messages that *causally precede* a claim. This notion corresponds closely to non-injective agreement.

As mentioned in the introduction, Roscoe introduces in [27] intensional specifications. These can be viewed as authentication of communications. The notion of injectivity does not seem to play a role in Roscoe's definition. Besides further research by Roscoe et al, there have been few attempts at formalising intensional forms of authentication. A notable exception is the definition of authentication by Adi, Debbabi and Mejri in [2]. Their authentication property requires a strict order on the messages. Furthermore, injectivity of the runs is required.
The authentication definition of [2] differs from injective synchronisation on two main points. First, it has a parameter, consisting of the set of communications to be authenticated. In our work, this parameter is fixed: it is defined as the set of communications that causally precede the claim event. We argue that this choice results in the strongest possible form of authentication. If the parameter is chosen to be a proper subset of the causally preceding communications set, it can be shown that the authentication is strictly weaker than injective synchronisation for the normal intruder model. On the other hand, if the parameter includes a communication that does not causally precede the claim, authentication will not hold for any protocol in all execution models that allow agents to abort runs, or that allow the network to delay messages. A second difference is that the authentication definition is strictly tailored for protocols involving two parties that communicate directly with each other. Thus, it cannot straightforwardly be used to express that two parties authenticate each other when they only communicate via e.g. a server. Also, it is not clear how it generalises to multi-party settings.

In [22], Lowe introduces an entire hierarchy of extensional specifications, corresponding to authentication of data. This builds on earlier work of [8] and [13], resulting in four different forms of authentication, viz. aliveness, weak agreement, non-injective agreement and injective agreement. On top of this, agreement on subsets of data items and recentness are considered (two topics we do not address here). In the course of time many subtly different extensional authentication properties have been proposed. Most of these derive directly from the work by Lowe.

In [4], Boyd proposes an alternative hierarchy of extensional goals for authentication protocols, which are oriented towards goals regarding established keys as well as the end results for the user. Similar to Gollmann, Boyd assumes that authentication is comprised of session key establishment and further communications being authenticated through the use of this key.

Authentication and agreement are also studied in [10] by Focardi and Martinelli in the context of the so-called GNDC scheme. In a process algebra extended with inference systems reflecting cryptographic actions, one can reduce reasoning about security properties with respect to any arbitrary environment to the analysis of the behavior of the security protocol in the most general environment. It is argued that the GNDC scheme is valid for establishing various security properties, in particular agreement (as well as its weaker variants). In [9], Focardi and Martinelli recast Lowe's notion of agreement in the GNDC scheme, and show that it is strictly stronger than two other notions of authentication: GNDC-authentication and spi-authentication from [1]. This implies that the latter two notions are also strictly weaker than injective synchronisation.

With respect to the analysis and verification of injectivity, we note that most approaches implement Lowe's definition of injectivity by way of a *counting* strategy: in any possible execution of the protocol the number of initiator runs may not exceed the number of corresponding responder runs. This counting argument can easily be used in a model-checking approach. Indeed, this is how injectivity is verified in the Casper/FDR tool chain [21,28]. Since it is only possible to model a finite and fixed number of scenarios, this approach will only provide an approximation of injectivity. Other approaches to the verification of security protocols, e.g. those based on logics (such as [5]) or on term rewriting (such as [12]) do not consider injectivity. The Strand Spaces [31] approach does not provide formal means to deal with injectivity. Instead, it is proposed to check authentication based on nonces, for example by using so-called *solicited authentication tests* as defined in [15]. These tests guarantee injectivity, based on nonce freshness. Authentication and injectivity are strongly connected in this view. In the HLPSL framework used by the Avispa tool set [16], injectivity of a data item is verified by ensuring that it is not replayed. If the correct data items are chosen, this can be used to verify that injective agreement holds.
We mention two examples of security protocol formalisms that deal explicitly with injectivity. Gordon and Jeffrey have devised a method [18] to verify injective correspondence relations for the π-calculus. This method allows for verification by type-checking of (injective) correspondences. A second example can be found in the protocol logic by Adi, Debbabi and Mejri [2], where pattern matching is used to express injectivity for two-party protocols. However, it is not clear how verification can be done efficiently.

6 Conclusions

In this section we summarise the main contributions of our research and discuss some directions for future work.

First of all, we have defined a general trace model for security protocols which allows for the definition of security properties in an intensional style. This model is not tied to a particular semantics, making it independent of e.g. the execution model of an agent and the intruder model. The starting point of the model is a role-based protocol description, where security claims are local to the protocol roles. Such a subjective security claim expresses what an agent may safely assume after having executed his part of the protocol. The main motivation for developing the model was to study synchronisation and agreement in an abstract setting, allowing us to pinpoint what the exact differences are. To this end, we formalised two forms of agreement: injective and non-injective agreement over all variables and all roles. Due to the uniform phrasing, the two notions of authentication can be distinguished easily: agreement allows that an intruder injects a (correct and expected) message before it is sent by the originator of the message. As for agreement, we provide both an injective and a non-injective variant of synchronisation.

Our definitions of synchronisation and agreement abstract away from the protocol and the semantic model as much as possible, e.g. they do not refer to the details of the message elements. Given a trace, we only need to have an equality relation between the contents of send and read events, and to know the ordering of the events in a protocol description, in order to be able to verify every form of authentication defined here. This contrasts with other definitions of authentication, where often much more information about the protocol and its semantics is required to verify authentication for a given trace. In fact, for our approach, the definitions do not even require a trace semantics or a full ordering on the events within a role. The definitions will also work with the partially-ordered structures of the Strand Spaces model of [31], but also with the preorder on the events of a role of the AVISPA model in [16]; the only requirement on the role event order is that each event must have a finite set of preceding events.
From the definitions of synchronisation and agreement, we construct a hierarchy of authentication properties depicted in Figure 2. We show that with respect to the Dolev-Yao intruder model, injective synchronisation is strictly stronger than injective agreement.

For the abstract model, we have only included those elements that are necessary to explain synchronisation. If this model is extended to a full semantics, as we did in e.g. [7], formal proofs that security protocols satisfy synchronisation can be constructed.

Theorem 2 states that, for a large class of security protocol models, injectivity of authentication protocols is easy to verify, once synchronisation has been established. Until now, injectivity and authentication have been strongly connected. Our new results establish that it suffices to verify the non-injective variant of synchronisation. Verifying injectivity is a simple and separate task, which does not depend on any specific (data) model.
For our injectivity result, we did not choose a specific security protocol model. Instead, as already mentioned, we have characterised a class of models in which Theorem 2 holds. This class contains nearly all models found in the literature, such as the Strand Spaces model, Casper/FDR without time, and term rewrite systems [31,21,12], as well as many models that allow for non-linear (branching) protocol specifications. These models share the following properties:

(i) Multiple instances of the protocol are truly independent. They do not share variables, memory, or time.
(ii) The intruder has the ability to duplicate messages, as holds, for example, in the standard Dolev-Yao intruder model.

The question arises whether the theorem also holds in an intruder-less model. This is in fact the case, but of less interest, because injectivity always holds for synchronising or agreeing protocols when there is no intruder.
Automated verification of the *LOOP* property can be implemented easily. We are currently working on an extension of the Scyther tool [6]. The algorithm is an instance of the reachability problem in a finite acyclic graph, and therefore has linear complexity.

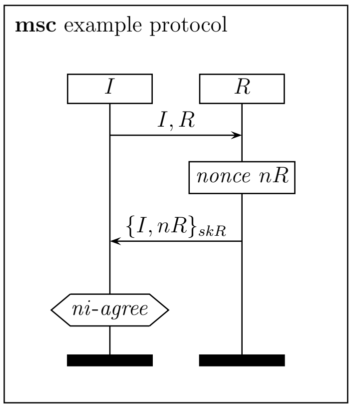

Fig. 9. A unilateral agreement protocol, with *LOOP*, but not injective.

Almost all correct authentication protocols in the literature satisfy *NI-SYNCH* as well as *LOOP*. It seems that *LOOP* is a necessary condition for injectivity. We know that this holds for the Dolev-Yao intruder model. However, for peculiar intruder models, *LOOP* is not a necessary condition for injectivity.
In the models where *LOOP* is also a necessary condition for injectivity, our

results imply a minimum number of messages in a multi-party authentication protocol. We will investigate this in future work.
The *LOOP* property guarantees injectivity for synchronising protocols. This raises the question whether there is a similar property to show injectivity of agreeing protocols. It can be seen from the example in Figure 9, that *LOOP* does not suffice to guarantee injectivity. The protocol satisfies the loop property for the claim role, and the protocol satisfies non-injective agreement, but not injective agreement. Finding an alternative for agreeing protocols is an interesting challenge for future research.

References

[1] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999.

[2] K. Adi, M. Debbabi, and M. Mejri. A new logic for electronic commerce protocols. *Theoretical Computer Science*, 291:223–283, 2003.

[3] A. Armando, D. Basin, Y. Boichut Y. Chevalier, L. Compagna, L. Cuellar, P.H. Drielsma, P. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proceedings of CAV'2005*, LNCS 3576, pages 281–285. Springer-Verlag, 2005.

[4] C. Boyd. Towards extensional goals in authentication protocols. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[5] M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8:16–36, 1990.

[6] C.J.F. Cremers. Scyther documentation, 2004. `http://www.win.tue.nl/~ccremers/scyther`.

[7] C.J.F. Cremers and S. Mauw. Operational semantics of security protocols. In S. Leue and T. Systä, editors, *Scenarios: Models, Transformations and Tools, International Workshop, Dagstuhl Castle, Germany, September 7-12, 2003, Revised Selected Papers*, volume 3466 of *LNCS*. Springer, 2005.

[8] W. Diffie, P.C. van Oorschot, and M.J. Wiener. Authentication and authenticated key-exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[9] R. Focardi, R. Gorrieri, and F. Martinelli. A comparison of three authentication properties. *Theor. Comput. Sci.*, 291(3):285–327, 2003.

[10] R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *World Congress on Formal Methods (1)*, pages 794–813, 1999.

[11] International Organization for Standardization. Information technology - security techniques - entity authentication - part 1: General model. ISO/IEC 9798-1, September 1991.

[12] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Conference on Automated Deduction*, pages 271–290. LNAI 1831, 2000.

[13] D. Gollmann. What do we mean by entity authentication. In *Proc. Symposium on Research in Security and Privacy*, pages 46–54. IEEE, 1996.

[14] L. Gong. Variations on the themes of message freshness and replay—or the difficulty of devising formal methods to analyze cryptographic protocols. In *Proceedings of the Computer Security Foundations Workshop VI*, pages 131–136. IEEE Computer Society Press, 1993.

[15] J.D. Guttman and F.J. Thayer. Authentication tests and the structure of bundles. *Theor. Comput. Sci.*, 283(2):333–380, 2002.

[16] FET Open Project IST-2001-39252. AVISPA: Automated validation of internet security protocols and applications, 2003. `http://www.avispa-project.org/`.

[17] ITU-TS. *Recommendation Z.120: Message Sequence Chart (MSC).* ITU-TS, Geneva, 1999.

[18] A. Jeffrey and A. Gordon. Typing One-to-One and One-to-Many Correspondences in Security Protocols. In Mitsuhiro Okada, Benjamin C. Pierce, Andre Scedrov, Hideyuki Tokuda, and Akinori Yonezawa, editors, *Proc. ISSS '02*, pages 418–434. LNCS 2514, 2002.

[19] S.T. Kent. Encryption-based protection for interactive user/computer communication. In *SIGCOMM '77: Proceedings of the fifth symposium on Data communications*, pages 5.7–5.13, New York, NY, USA, 1977. ACM Press.

[20] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Proc. TACAS '96*, pages 147–166. LNCS 1055, 1996.

[21] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proc. CSFW '97, Rockport*, pages 18–30. IEEE, 1997.

[22] G. Lowe. A hierarchy of authentication specifications. In *Proc. CSFW '97, Rockport*, pages 31–44. IEEE, 1997.

[23] S. Malladi, J. Alves-Foss, and R. Heckendorn. On preventing replay attacks on security protocols. In *Proc. International Conference on Security and Management*, pages 77–83. CSREA Press, 2002.

[24] C.J. Mitchell and L. Chen. Comments on the s/key user authentication scheme. *SIGOPS Oper. Syst. Rev.*, 30(4):12–16, 1996.

[25] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21:120–126, 1978.

[26] G.J. Popek and C.S. Kline. Encryption and secure computer networks. *ACM Comput. Surv.*, 11(4):331–356, 1979.

[27] A.W. Roscoe. Intensional Specifications of Security Protocols. In *Proc. CSFW '96*, pages 28–38. IEEE, 1996.

[28] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols.* Addison Wesley, 2000.

[29] D. Song. Athena: a new efficient automatic checker for security protocol analysis. In *Proc. CSFW '99, Mordano*, page 192. IEEE, 1999.

[30] Security protocols open repository. `http://www.lsv.ens-cachan.fr/spore`.

[31] F.J. Thayer, J.C. Herzog, and J.D. Guttman. Strand spaces: Why is a security protocol correct. In *Proc. Symposium on Security and Privacy, Oakland*, pages 160–171. IEEE, 1998.