A framework for compositional verification of security protocols

Suzana Andova [a,1], Cas Cremers [c,4], Kristian Gjøsteen [b,2], Sjouke Mauw [d], Stig F. Mjølsnes [a], Saša Radomirović [d,*,1,3]

[a] *Dept. of Telematics, NTNU, N-7491 Trondheim, Norway.*

[b] *Dept. of Mathematical Sciences, NTNU, N-7491 Trondheim, Norway.*

[c] *Dept. of Computer Science, ETH Zürich, 8092 Zürich, Switzerland.*

[d] *Université du Luxembourg, Faculté des Sciences, de la Technologie et de la Communication, 6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg.*

Abstract

Automatic security protocol analysis is currently feasible only for small protocols. Since larger protocols quite often are composed of many small protocols, compositional analysis is an attractive, but non-trivial approach.

We have developed a framework for compositional analysis of a large class of security protocols. The framework is intended to facilitate automatic as well as manual verification of large structured security protocols. Our approach is to verify properties of component protocols in a multi-protocol environment, then deduce properties about the composed protocol. To reduce the complexity of multi-protocol verification, we introduce a notion of protocol independence and prove a number of theorems that enable analysis of independent component protocols in isolation.

To illustrate the applicability of our framework to real-world protocols, we study a key establishment sequence in WiMAX consisting of three subprotocols. Except for a small amount of trivial reasoning, the analysis is done using automatic tools.

Key words: compositionality, security protocols, automatic verification, WiMAX, security properties, authentication, confidentiality, semantics

1 Introduction

Security protocols are a crucial component of many contemporary applications. Their security is however very difficult to assess for humans, mainly due to the vast number of attack options available to an adversary. To deal with this complexity, a structured approach is needed. Starting from abstract protocols, formal methods faciliate the systematic detection of attacks or the generation of a proof of correctness. Automating this process in order to minimize the risk of human error is one of the major goals in security protocol analysis.

Automatic protocol verification is, in general, a complex task even for short protocols. The time needed for verification of a protocol using modern methods employed by state of the art tools such as Scyther [11] or AVISPA [4] is still exponential with respect to the number of messages. Consequently, automatic verification of large protocols is currently infeasible. In this paper, we attempt to narrow the gap between small, academic protocols and large, industrial protocols by taking advantage of compositional verification.

Large protocols are usually built from structured components. They typically consist of several (optional) protocols composed in parallel, or a sequential composition of a key establishment protocol and a secure data transfer protocol that uses the key. For instance, IPSec, SET, and WiMAX have all been designed with such a principle in mind. A compositional approach to the design and analysis of security protocols is therefore natural and expected to reduce the complexity of the analysis of the large protocol to the order of the complexity of the analysis of the largest component. This could be achieved by first verifying properties of the components in isolation and then using the results to deduce properties of the composed protocol. However, as no generic compositionality results are known, further assumptions are needed to facilitate this type of reasoning.

We illustrate the non-triviality of protocol composition by means of the well-known Needham-Schroeder-Lowe (*NSL*) public key authentication protocol [32,

* Corresponding author. Phone: (+352) 46 66 44 5484, Fax: (+352) 46 66 44 5500
Email addresses: suzana@item.ntnu.no (Suzana Andova), cremersc@inf.ethz.ch (Cas Cremers), kristian.gjosteen@math.ntnu.no (Kristian Gjøsteen), sjouke.mauw@uni.lu (Sjouke Mauw), sfm@item.ntnu.no (Stig F. Mjølsnes), sasa.radomirovic@uni.lu (Saša Radomirović).
[1] This work was partially carried out during the tenure of an ERCIM Fellowship.
[2] Supported in part by the Norwegian Research Council project 158597 NTNU Research Programme in Information Security.
[3] Supported in part by a Centre de Recerca Matemàtica Postdoctoral Fellowship.
[4] Supported in part by the Hasler Foundation project 2071.

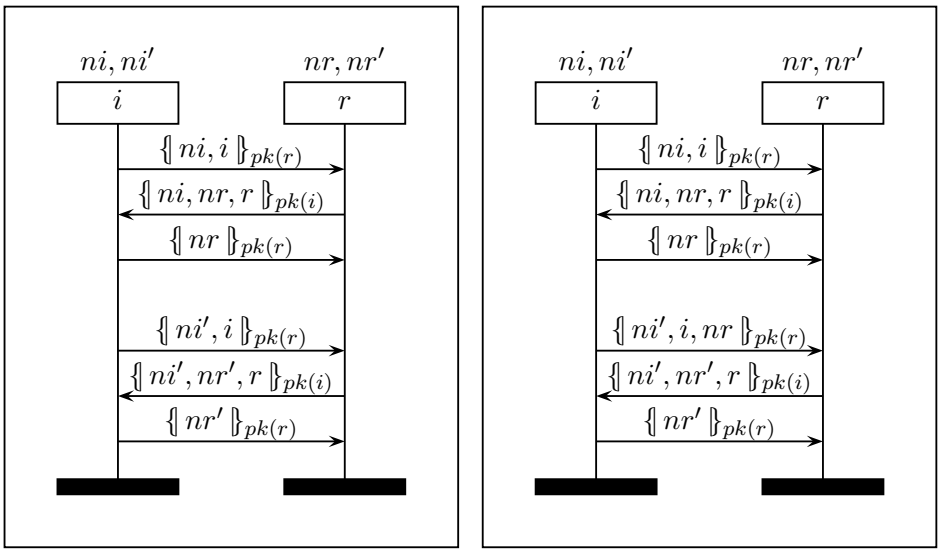


Fig. 1. Repeated *NSL* protocol: incorrect and correct chaining.

37]. In isolation, it satisfies even the strongest forms of authentication, such as agreement and synchronization [16]. However, when sequentially composing this protocol with itself (see the left drawing in Figure 1), authentication is not preserved. The reason is that the initiator i may successfully finish his run of the composed protocol, while the responder r possibly never executed the second half of the protocol. This is because the second half of the initiator's run may match to the first half of a different run of the responder. This authentication problem is illustrated in Figure 2. Here we see agent A executing the initiator role i and agent B executing two different runs of the responder role r. The intruder links the messages as indicated. Run $A(i)$ and run $B(r)\sharp 2$ will agree on the values of ni, and nr, but not on the values of ni' and nr', since these last two values are not communicated between these two runs. In a similar way, it is clear that run $A(i)$ and run $B(r)\sharp 1$ do not agree on the supposedly shared nonces.

This problem is solved in the right drawing in Figure 1 by chaining the two protocols. A nonce from the first instance of *NSL* is repeated as payload in the second instance. In this way the two protocols become linked and the chained protocol satisfies authentication. The authentication problem from Figure 2 is now impossible.

Even though it is well known that the composition of secure protocols is in general not secure [2,13,25,31] and compositionality has been recognised as one of the open challenges for security protocol analysis [12,36], the vast majority of formalisms and tools for security protocols have only addressed single-protocol (i.e. non-composed) analysis and verification. Early work on identifying and addressing the problem includes [38]. An initial attempt within the Strand Spaces model [42] has led to some theoretical results about compositionality. The Strand Spaces approach is similar to the one taken here in that both at-

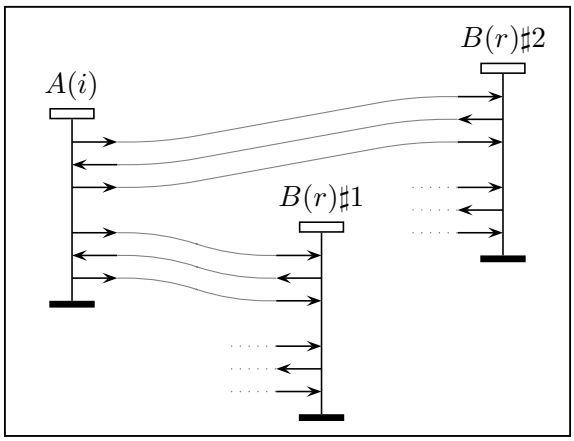


Fig. 2. Authentication problem in incorrectly chained NSL protocol.

tempt to identify the abstract properties two protocols need to satisfy in order to be securely composable. However, this work significantly improves upon the Strand Spaces approach in terms of efficiency in verifying composed protocols and by considering sequential composition, which was absent in the Strand Spaces model. One of the recent significant developments in compositional protocol analysis is Protocol Composition Logic (PCL) [17, 18]. It provides support for compositional reasoning, and has been applied in a number of case studies, including the verification of the TLS and IEEE 802.11i protocols [24] and contract signing protocols [5]. While the PCL approach is quite general, it cannot, in contrast to the present approach, be easily automated.

In this paper, we develop a framework to verify security properties of protocols that are composed from several smaller protocols. We prove several theorems concerning the deduction of properties of a sequential composition of two protocols from properties these protocols have when running together in a multi-protocol environment. With these theorems, we reduce the analysis of a sequential composition to the analysis of the component protocols running together.

Analysing several protocols in a multi-protocol environment is, in general, no easier than analysing their sequential composition. In order to make automatic analysis feasible, we introduce the notion of protocol set independence, where ciphertexts, signatures, and message authentication tags originating in one protocol set will never be accepted by the other protocol set and vice versa. This notion allows us to prove several theorems regarding the deduction of properties of protocols running together in a multi-protocol environment from properties these protocols have when running in isolation.

Verifying independence itself is non-trivial, therefore we need the notion of strong independence, where the forms of ciphertexts, message authentication tags, and signatures in the two protocol sets are sufficiently different to prevent confusion. Strong independence can be easily verified at the syntactical level,

and implies independence. We show that through common design strategies for security protocols in current use, strong independence will be satisfied. Note that different protocols can use the same cryptographic keys and still be both, independent and strongly independent.

The model we use is based on the operational semantics for security protocols defined in [15]. In contrast to other approaches, in which only singular protocols are considered, this model provides a semantics of protocols in a multi-protocol setting. This makes it a good starting point for compositional verification, since, as indicated, the problem of proving correctness of a composed protocol can be translated into the problem of proving correctness of the components in a multi-protocol setting comprising the components themselves.

To show the applicability of our work, we perform a case study. We have chosen to focus on the IEEE 802.16 standard, also known as WiMAX. This standard specifies the air interface of wireless access systems featuring a security sublayer intended to protect network operators from theft of service and provide confidentiality to subscribers. WiMAX features a security sublayer consisting of several subprotocols for authentication, key management, and secure communication. This makes WiMAX well suited for an analysis in our framework. Our verification is completely tool-supported, except for some trivial reasoning and theorem application.

Overview of the paper

We start off by giving a brief description of the security protocol model and security properties used in Section 2. In Section 3, we develop a framework for compositional reasoning about security protocols, and prove a number of compositionality theorems. We show how the developed theory can be applied in practice by performing a case study on key management protocols in the security sublayer of WiMAX in Section 4. Related work is discussed in Section 5, and we draw conclusions and discuss future work in Section 6.

2 Security Protocols and Their Semantics

In this section we describe an existing formal framework for modeling security protocols, and extend it with notions relevant for compositional reasoning.

We begin by giving a brief overview of the model in Section 2.1 before describing the full technical details in Sections 2.2 and 2.3. The model presented

here is based on the model defined in [15]. Readers who are familiar with the basic model may skip to Section 2.4 on page 14, as the only change is the introduction of parameters for protocols.

In Sections 2.4 and 2.5 we further extend the model with features not present in the basic model defined in [15], namely trace restrictions (similar to preconditions in PCL and elsewhere), satisfiability predicates, and new security notions.

2.1 Overview

The basic entities in our framework are *role specifications*. Every role specification consists of a sequence of uniquely *labeled events* describing the messages an agent shall send and receive, when it executes the role specification, as well as certain *security claims*. The role specification includes *constants* which roughly correspond to nonces, *variables* which store values read from the network, and *parameters* which represent input.

A *protocol* is a collection of role specifications that communicate by sending and receiving messages. More precisely, a protocol is a partial function, mapping role names to role specifications. A *run* is an execution of a role specification by an *agent*. Communication between runs is asynchronous and is modeled by agents reading messages from and writing messages to a shared input/output buffer (by executing read and send events). As the buffer is completely under the control of the adversary, according to the standard Dolev-Yao intruder model [19], we identify the buffer with the intruder knowledge. The actual behavior of the entire system, consisting of the intruder and a set of agents executing a number of runs, is encoded in the *traces* of the system. In some situations, we are not interested in all possible traces but in a subset of traces that have a certain property; for instance, the subset of traces whose input values are secret. In that case, we talk about *trace restriction*.

Security properties in our framework are local to a role and are described by the *claim* events in the role specifications. Every claim event in a trace results in a statement about the trace that may or may not be true. In this paper, we focus on three security properties: *secrecy*, *authentication*, and *session key establishment*. A secrecy claim event is essentially the statement that something never enters the adversary's knowledge, as determined by the trace. Authentication is captured by the notion of *synchronization*. A synchronization claim event translates into the statement that there are runs for the other protocol roles in the trace with read and send events that match this run's send and read events exactly, both in content and in order. Our notion of *session keys* is that a session key is secret and identifies a protocol session, in the sense

that there is exactly one execution of every protocol role sharing the session key.

2.2 Security Protocol Specification

Let $\mathcal{ID}$ be a set of *identifiers*, $\mathcal{R}$ a set of *role names* or *roles* for short, and $\mathcal{F}$ a set of (global) functions. There are three types of identifiers: *constants*, *variables*, and *parameters*. Constants include the general notion of nonces, and we will informally refer to some constants as nonces. Concatenation or tupling of terms is written as (x, y). Encryptions of a term x with a term y are denoted by $\{\!|\, x \,|\!\}_y$. *Role terms* can be considered as templates for messages that are read or sent by the agents. The set of role terms is defined as:

$$\begin{aligned} \mathit{RoleTerm} ::= \mathcal{ID} \mid \mathcal{R} \mid \mathcal{F}(\mathit{RoleTerm}^*) \\ \mid (\mathit{RoleTerm}, \mathit{RoleTerm}) \mid \{\!|\, \mathit{RoleTerm} \,|\!\}_{\mathit{RoleTerm}} \end{aligned}$$

Terms that have been encrypted with a term, can only be decrypted by its inverse term which is either the same term (for symmetric encryption) or the inverse key (for asymmetric encryption). We use k^{-1} to denote the inverse key of a key k. In this work, functions from $\mathcal{F}$ are only used to construct long-term keys, such as $pk(r)$, $sk(i)$, $k(x, y)$. Short term session keys are represented by constants. In the remainder of the paper x, y, z range over $\mathit{RoleTerm}$, and c, d over the $\mathcal{ID}$ set.

Example 1. The first message sent by the initiator in the *NSL* protocol is denoted by $\{\!|\, ni, i \,|\!\}_{pk(r)}$, where $ni \in \mathcal{ID}$ is a constant, $i, r \in \mathcal{R}$ are role names, and $pk() \in \mathcal{F}$.

We say that x_1 is a *subterm* of x_2 if $x_1 \sqsubseteq x_2$, where $\sqsubseteq$ is the smallest transitive relation satisfying the following rules, for all terms x_1, x_2:

$$x_1 \sqsubseteq x_1, \quad x_1 \sqsubseteq (x_1, x_2), \quad x_2 \sqsubseteq (x_1, x_2), \quad x_1 \sqsubseteq \{\!|\, x_1 \,|\!\}_{x_2}, \quad x_2 \sqsubseteq \{\!|\, x_1 \,|\!\}_{x_2}.$$

For a given set of *labels* $\mathcal{L}$ and a set of *claims* *Claim* we define the set of *events* $\mathcal{E}$ as:

$$\begin{aligned} \mathcal{E} = \Big\{ \mathit{create}_\ell(r), \mathit{send}_\ell(r, r', x), \mathit{read}_\ell(r', r, x), \mathit{claim}_\ell(r, c\,[, x]), \mathit{end}_\ell(r) \;\Big| \\ \ell \in \mathcal{L}, r, r' \in \mathcal{R}, x \in \mathit{RoleTerm}, c \in \mathit{Claim} \Big\} \end{aligned}$$

The labels ℓ extending the events are needed to disambiguate multiple occurrences of an event in protocol specifications (e.g. when composing two instances of the NSL protocol). A second use of these labels is to express which

send and read events are supposed to correspond (e.g. in NSL the first message sent by the initiator is linked to the first message received by the responder).

Event $send_\ell(r, r', x)$ denotes the sending of message x by r, apparently to r'. Likewise, $read_\ell(r', r, x)$ denotes the reception of message x by r', apparently sent by r. We interpret role terms of the form $\{\!|\, u \,|\!\}_v$ in a *send* event as encryption with symmetric or public encryption keys, or signing with private signing keys. In a *read* we interpret it as decryption with symmetric or private encryption keys, or verification with public signing keys. An agent can encrypt or decrypt a term only when it has the relevant key in its knowledge. Event $claim_\ell(r, c\,[, x])$ expresses that r upon execution of this event expects the security property associated with the claim c to hold with optional argument x. A claim event is always local to a role, and does not imply that other roles expect the security property associated with the claim c to hold for them. Events $create_\ell(r)$ and $end_\ell(r)$ are used to signal the start and end of the role.

Example 2. The first send event of the initiator in the *NSL* protocol is denoted by $send_{\ell_1}(i, r, \{\!|\, ni, i \,|\!\}_{pk(r)})$, where $ni \in \mathcal{ID}$ is a constant and ℓ_1 is some label. The first read event of the responder in the NSL protocol is denoted by $read_{\ell_1}(i, r, \{\!|\, ni, i \,|\!\}_{pk(r)})$, where $ni \in \mathcal{ID}$ is a variable.

A *role specification* is a pair $(elist, type)$ where $elist \in \mathcal{E}^*$ is a list of events and $type : \mathcal{ID} \to \{const, param, variable\}$ is a function that assigns types to the identifiers that appear in *elist*. We require that there is only one *create* and one *end* in the event list, and that they start and terminate the list. Furthermore, we require that the role names in the *create* and *end* events are the same, and they match the role names that appear in *claim* events and the sender and recipient, respectively, in *send* and *read* events. This is the specification's *role name*. The set of all role specifications is denoted by *RoleSpec*.

Note that only in the context of a role specification rs can we talk about the set of variables or parameters. For a role specification $rs = (elist, type)$, we write $var_{rs}(x)$ for the set of identifiers that appear in role term x and are considered variables in the role specification.

A *protocol* is a partial mapping of role names to role specifications, i.e. $\mathcal{R} \to RoleSpec$. We say that r is a role in protocol P if $r \in dom(P)$, the domain of P. If r is a role in protocol P and ℓ is a label of an event in the event list of r then we write $\ell \in P(r)$. We extend this notation in the obvious way to $\ell \in P$. By $\mathcal{ID}(P)$ we denote the set of all identifiers that appear in protocol P. The universe of protocols is denoted by *Prot*.

For a protocol P, we require that all labels are unique, except for the labels of corresponding read and send events which have to be identical. For a set of protocols Π, we require that a label is used in at most one protocol.

We define a relation $\prec'$ on the events of a protocol as the union of the obvious event orders on the role specifications. We extend this relation with all pairs of identically labeled *send* and *read* events so that such *send* events always precede the corresponding *read* events. The partial order $\prec$ is the transitive closure of $\prec'$ and represents causality preorder.

Example 3. The following example specifies the *NSL'* protocol, which is the bottom right subprotocol in Figure 1. Notice the parameter nr and the fact that ni' is considered a constant by role i, whereas it is a variable for role r.

$$\begin{aligned}
NSL'(i) &= (create_1(i) \cdot send_2(i, r, \{\!|\, ni', i, nr \,|\!\}_{pk(r)}) \cdot \\
&\quad read_3(r, i, \{\!|\, ni', nr', r \,|\!\}_{pk(i)}) \cdot send_4(i, r, \{\!|\, nr' \,|\!\}_{pk(r)}) \cdot end_5(i), \\
&\quad \{nr \mapsto param, ni' \mapsto const, nr' \mapsto variable\}) \\
NSL'(r) &= (create_6(r) \cdot read_2(i, r, \{\!|\, ni', i, nr \,|\!\}_{pk(r)}) \cdot \\
&\quad send_3(r, i, \{\!|\, ni', nr', r \,|\!\}_{pk(i)}) \cdot read_4(i, r, \{\!|\, nr' \,|\!\}_{pk(r)}) \cdot end_7(r), \\
&\quad \{nr \mapsto param, ni' \mapsto variable, nr' \mapsto const\})
\end{aligned}$$

2.3 Runs and Traces

In this section we describe how, through instantiation, an abstract role specification can be transformed into an execution of a role, which we call a *run*. Furthermore, we define how the interleaved operation of a collection of runs defines the *traces* of a system.

Run terms model the actual messages sent in a protocol. Since the run terms are instantiations of role terms they are defined similarly. Let $Runid$ be a set of *run identifiers*, $\mathcal{IT}$ a set of *intruder-generated* run terms and $\mathcal{A}$ a set of *agent names* which is a disjoint union of a set of trusted and a set of untrusted agents, $\mathcal{A}_T$ and $\mathcal{A}_U$ respectively. The set of run terms is defined as:

$$\begin{aligned}
RunTerm ::= \mathcal{A} \mid \mathcal{F}(RunTerm^*) \mid \mathcal{ID}\sharp Runid \mid \mathcal{IT} \mid \\
(RunTerm, RunTerm) \mid \{\!|\, RunTerm \,|\!\}_{RunTerm}
\end{aligned}$$

The run terms of the form $\mathcal{ID}\sharp Runid$ and the terms in $\mathcal{IT}$ are called *nonce run terms*. The subterm relation $\sqsubseteq$ on run terms is defined similarly to the subterm relation on role terms. Since it is clear from the context which one is used, we allow the same notation for both relations. In the remainder of the paper t, u, v range over $RunTerm$.

A role term is turned into a run term when abstract role names are replaced by concrete agent names, and constants are made unique by extending them

with a run identifier. This is done by means of an *instantiation*, which is a triplet (rid, ρ, σ), where $rid \in Runid$, ρ is a partial function from role names to agent names, and σ is a partial function from identifiers to run terms. We denote the set of all possible instantiations by $Inst$.

In the context of some role specification rs with type function $type$, an instantiation $inst = (rid, \rho, \sigma)$ turns a role term x into a run term, if ρ is defined for every role name that appears in x and $var_{rs}(x) \subseteq dom(\sigma)$. For any $f \in \mathcal{F}$ and role terms $x_1, \ldots, x_n \in RoleTerm$, instantiation is defined recursively by:

$$inst(x) = \begin{cases} \rho(r) & \text{if } x \equiv r \in \mathcal{R} \\ c\sharp rid & \text{if } x \equiv c \in \mathcal{ID} \wedge type(c) = const \\ \sigma(x) & \text{if } x \in \mathcal{ID} \wedge type(x) \in \{param, variable\} \\ f(inst(x_1), \ldots, inst(x_n)) & \text{if } x \equiv f(x_1, \ldots, x_n) \\ (inst(x_1), inst(x_2)) & \text{if } x \equiv (x_1, x_2) \\ \{\!|\, inst(x_1) \,|\!\}_{inst(x_2)} & \text{if } x \equiv \{\!|\, x_1 \,|\!\}_{x_2} \end{cases}$$

If an instantiation cannot be applied because $var_{rs}(x) \not\subseteq dom(\sigma)$, we say that x has *free variables* in this context.

Example 4. If we apply instantiation $(42, \{i \mapsto a, r \mapsto b\}, \{ni \mapsto ni\sharp 41\})$ to the contents of the first send event of the responder in the *NSL* protocol $\{\!|\, ni, nr, r \,|\!\}_{pk(i)}$, we obtain $\{\!|\, ni\sharp 41, nr\sharp 42, b \,|\!\}_{pk(a)}$,

Instantiations are essential ingredients to define the notion of a *run* of a role. A *run* of a role specification $rs = (elist, type)$ is a pair $(inst, elist')$, where $inst \in Inst$ and $elist'$ is a suffix of $elist$. In this definition we express that one is mainly interested in the current state of an agent executing a role. We model this dynamic aspect by requiring that the list $elist' \in \mathcal{E}^*$ contains the remaining events in the role specification, and not the complete role specification. The instantiation $inst$ contains the actual values of the variables and parameters, as well as the agent names expected to execute the other protocol roles. The set of all runs is denoted by $Runs$. A *run event* is a pair $(inst, ev) \in Inst \times \mathcal{E}$. These are the events that can be observed when executing a system. A system's behavior is represented by a sequence of run events, which we call a *trace*. The universe of traces is denoted by $Traces$.

Let P be a protocol with a role specification $rs = (elist, type)$, and let $inst = (rid, \rho, \sigma)$ be an instantiation. The pair $(inst, elist)$ is an *initial run* for rs if and only if $dom(\rho) = dom(P)$ and $dom(\sigma) = type^{-1}(param)$ (σ is defined for all role parameters, specifying the run's input). The set of all initial runs for

all roles of a protocol P is denoted by $runsof(P)$. For a protocol set Π, we let

$$runsof(\Pi) = \bigcup_{P \in \Pi} runsof(P).$$

Example 5. An initial run of the initiator of the *NSL'* protocol from Example 3 is $((42, \{i \mapsto a, r \mapsto b\}, \{nr \mapsto ni\sharp 41, nr' \mapsto \bot\}), create_1(i) \cdot send_2(i, r, \{\!|\, ni', i, nr \,|\!\}_{pk(r)}) \cdot read_3(r, i, \{\!|\, ni', nr', r \,|\!\}_{pk(i)}) \cdot send_4(i, r, \{\!|\, nr' \,|\!\}_{pk(r)}) \cdot end_5(i))$. Notice that nr is the only parameter (and thus must be initialized), that variable nr' has no initial value and that ni' is a constant.

For a protocol set Π we consider a system with a number of runs (communicating with each other) executed by agents in presence of an intruder. We assume a standard Dolev-Yao model, in which the intruder has complete control over the communication network. The knowledge of the intruder, denoted by M, is a subset of run terms. He can decrypt messages if he knows the appropriate decryption key, and he can construct messages from his knowledge set. We express this by requiring that M is closed, that is:

$$\begin{aligned} \forall_{u,v \in M}(u, v) \in M &\Rightarrow \{\!|\, u \,|\!\}_v \in M \\ \forall_{u,v} \{\!|\, u \,|\!\}_v, v^{-1} \in M &\Rightarrow u \in M \\ \forall_{u,v}(u, v) \in M &\Leftrightarrow u, v \in M \end{aligned}$$

The closure $\overline{M}$ of a set of run terms M is the smallest closed superset of M.

Due to the dynamic behavior of the system, the intruder knowledge increases during the execution. We assume that the initial knowledge M_0 of the intruder can be derived from the protocol and the context (e.g. the public keys of all agents and the secret keys of all compromised agents). We require that $\mathcal{IT} \subseteq M_0$. The derivation of the initial intruder knowledge from the protocol specification is treated in detail in [14].

The behavior of the system is defined as a transition relation between system states. Every state is determined by an intruder knowledge set M containing run terms (which is also used to model an asynchronous communication between agents), and a set F containing all active runs. We denote by $runids(F)$ the set of all run identifiers that appear in F. Every transition is labeled with a run event $(inst, ev) \in Inst \times \mathcal{E}$.

The derivation rules for the system are given in Table 1. We denote by $F[x/y]$ the set obtained from F when x replaces y. Note that from run events used to label transitions, one can uniquely determine role specifications. All instantiations that appear in a rule are applied in the context of this role specification.

The *create* rule expresses that a new run can only be created if its run identifier has not been used yet. The *end* and *claim* rules express that these events can always be executed. Recall that $\overline{M}$ denotes the closure of the set M. The *send*

rule states that if a run executes a send event, the sent message (obtained by instantiating a role term in the role specification context determined by the run event) is added to the intruder knowledge and the executing run proceeds to the next event.

The *read* rule determines when a read event can be executed, with the help of a match predicate defined as follows:

$$\begin{aligned} \mathit{Match}(\mathit{inst}, m, t, \mathit{inst}') \iff & \mathit{inst} = (\mathit{rid}, \rho, \sigma) \wedge \mathit{inst}' = (\mathit{rid}, \rho, \sigma') \wedge \sigma \subseteq \sigma' \wedge \\ & \mathit{dom}(\sigma') = \mathit{dom}(\sigma) \cup \mathit{var}_{rs}(m) \wedge \mathit{inst}'(m) = t. \end{aligned}$$

The match predicate decides if an incoming message t can be matched against a pattern specified by a role term m. With respect to the first instantiation *inst*, the pattern may contain free variables. The idea is that the second instantiation inst' extends the first instantiation by assigning values to the free variables such that the incoming message equals the instantiated role term. Note that the run event determines role specification rs.

Example 6. We have $\mathit{Match}(\mathit{inst}, m, t, \mathit{inst}')$ for $\mathit{inst} = (42, \{i \mapsto a, r \mapsto b\}, \{nr \mapsto \bot\})$, $m = \{\!|\, ni, nr, r \,|\!\}_{pk(i)}$, $t = \{\!|\, ni\sharp 42, nr\sharp 12, b \,|\!\}_{pk(a)}$, and $\mathit{inst}' = (42, \{i \mapsto a, r \mapsto b\}, \{nr \mapsto nr\sharp 12\})$. This models the first receive event of agent a executing the initiator role of *NSL* in run 42. The symbol $\bot$ means that no value is assigned.

A state transition is the conclusion of an application of one of these rules. In this way, starting from the initial state $\Sigma_0 = \langle M_0, \emptyset \rangle$, where M_0 refers to the initial intruder knowledge, we can derive all possible behaviors of a system executing a protocol set Π.

We define the *set of traces* generated by the above derivation rules as a subset of *Traces*. Let $\alpha \in \mathit{Traces}$ be a trace of length $|\alpha| = n$, and denote by α_i the ith run event in α (starting with 0). Then α is a valid trace for the system if there exist states $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ such that $\Sigma_0 \xrightarrow{\alpha_0} \Sigma_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{n-1}} \Sigma_n$ is a valid derivation. We denote the set of all valid traces for the protocol set Π by $\mathit{Tr}(\Pi)$. When we consider the trace set $\mathit{Tr}(\{P\} \cup \Pi)$ we say that P runs in the context of Π.

We reconstruct state information from a trace as follows. If α_i is a run event from trace α, then M_i^α is the intruder knowledge component M of the state right before the execution of α_i. Thus for all protocols P and traces $\alpha \in \mathit{Tr}(P)$, $M_0^\alpha = M_0$.

Next, we define a useful short hand. Let Π be a protocol set with $P \in \Pi$, and let $\alpha \in \mathit{Tr}(\Pi)$. A *cast* for P in α is a map $\mathit{cast} : \mathit{dom}(P) \to \mathit{Runid}$ such that for some fixed ρ, for every role $r \in \mathit{dom}(P)$ there is a run event $\alpha_i = ((\mathit{cast}(r), \rho, \cdot), \mathit{create}_\ell(r))$ with $\ell \in P$. Intuitively, for a trace α, a cast is

$$[create]\frac{run = (inst, create_\ell(r) \cdot elist) \in runsof(\Pi), inst = (rid, \rho, \sigma), rid \notin runids(F)}{\langle M, F\rangle \xrightarrow{(inst, create_\ell(r))} \langle M, F \cup \{(inst, elist)\}\rangle}$$

$$[end]\frac{run = (inst, end(r)) \in F,}{\langle M, F\rangle \xrightarrow{(inst, end_\ell(r))} \langle M, F[(inst, \varepsilon)/run]\rangle}$$

$$[send]\frac{run = (inst, send_\ell(m) \cdot elist) \in F}{\langle M, F\rangle \xrightarrow{(inst, send_\ell(m))} \langle \overline{M \cup \{inst(m)\}}, F[(inst, elist)/run]\rangle}$$

$$[read]\frac{run = (inst, read_\ell(m) \cdot elist) \in F, t \in M, Match(inst, m, t, inst')}{\langle M, F\rangle \xrightarrow{(inst', read_\ell(m))} \langle M, F[(inst', elist)/run]\rangle}$$

$$[claim]\frac{run = (inst, claim_\ell(r, c\,[, x]) \cdot elist) \in F}{\langle M, F\rangle \xrightarrow{(inst, claim_\ell(r, c\,[, x]))} \langle M, F[(inst, elist)/run]\rangle}$$

Table 1
Derivation rules.

an assignment of runs to roles, which expresses the possibility that these runs together form a session of the protocol. We denote the set of casts for P and α by $Cast(P, \alpha)$.

Example 7. We illustrate the concept of a trace by providing, in Figure 3, a possible trace of the *NSL'* protocol from Example 3. This trace consists of the execution of three runs. The first run is an instantiation of role i, with instantiation $(1, \{i \mapsto a, r \mapsto b\}, \{nr \mapsto u, nr' \mapsto \bot\})$, where a and b are agents, u is a nonce run term, and $\bot$ means that no value has been assigned yet. The second run instantiates role i as $(2, \{i \mapsto a, r \mapsto b\}, \{nr \mapsto v, nr' \mapsto \bot\})$, for nonce run term $v \neq u$. The third run is an instantiation of the responder role r, through $(3, \{i \mapsto a, r \mapsto b\}, \{nr \mapsto u, ni' \mapsto \bot\})$. Since runs 1 and 3 are instantiated such that they correspond, they can be executed up to completion. In contrast, run 2 is blocked.

After execution of this trace the intruder knowledge M is extended with the information contained in the four send events from the trace. Thus we have that M is equal to:

$$\overline{M_0 \cup \{\!|\, ni'\sharp 2, i, v \,|\!\}_{pk(r)}, \{\!|\, ni'\sharp 1, i, u \,|\!\}_{pk(r)}, \{\!|\, ni'\sharp 1, nr'\sharp 3, r \,|\!\}_{pk(i)}, \{\!|\, nr'\sharp 3 \,|\!\}_{pk(r)}\}}$$

There are two casts for the *NSL'* protocol in this trace: $\{i \mapsto 1, r \mapsto 3\}$ and $\{i \mapsto 2, r \mapsto 3\}$.

instantiation	event
$(1, \rho, \{nr \mapsto u, nr' \mapsto \bot\})$	$create_1(i)\cdot$
$(2, \rho, \{nr \mapsto v, nr' \mapsto \bot\})$	$create_1(i)\cdot$
$(2, \rho, \{nr \mapsto v, nr' \mapsto \bot\})$	$send_2(i, r, \{\!\lvert\, ni', i, nr \,\rvert\!\}_{pk(r)})\cdot$
$(1, \rho, \{nr \mapsto u, nr' \mapsto \bot\})$	$send_2(i, r, \{\!\lvert\, ni', i, nr \,\rvert\!\}_{pk(r)})\cdot$
$(3, \rho, \{nr \mapsto u, ni' \mapsto \bot\})$	$create_6(r)\cdot$
$(3, \rho, \{nr \mapsto u, ni' \mapsto ni'\sharp 1\})$	$read_2(i, r, \{\!\lvert\, ni', i, nr \,\rvert\!\}_{pk(r)})\cdot$
$(3, \rho, \{nr \mapsto u, ni' \mapsto ni'\sharp 1\})$	$send_3(r, i, \{\!\lvert\, ni', nr', r \,\rvert\!\}_{pk(i)})\cdot$
$(1, \rho, \{nr \mapsto u, nr' \mapsto nr'\sharp 3\})$	$read_3(r, i, \{\!\lvert\, ni', nr', r \,\rvert\!\}_{pk(i)})\cdot$
$(1, \rho, \{nr \mapsto u, nr' \mapsto nr'\sharp 3\})$	$send_4(i, r, \{\!\lvert\, nr' \,\rvert\!\}_{pk(r)})\cdot$
$(3, \rho, \{nr \mapsto u, ni' \mapsto ni'\sharp 1\})$	$read_4(i, r, \{\!\lvert\, nr' \,\rvert\!\}_{pk(r)})\cdot$
$(3, \rho, \{nr \mapsto u, ni' \mapsto ni'\sharp 1\})$	$end_7(r)\cdot$
$(1, \rho, \{nr \mapsto u, nr' \mapsto nr'\sharp 3\})$	$end_5(i)$

Fig. 3. Example trace of the *NSL*' protocol ($\rho = \{i \mapsto a, r \mapsto b\}$).

2.4 Trace restrictions

In Sections 2.2 and 2.3 we have described the semantics proposed by Cremers and Mauw in [15]. In this and the next section we define mechanisms for properly handling parameters, and we add several new security properties to the semantics, both important for protocol composition.

Note that there are essentially no restrictions on which values parameters take in the semantics. We interpret parameters as input to the protocols, and as such we need to specify where the protocol gets its input from. We do this on the level of protocol sets by specifying which protocols produce output and which protocols are allowed to use this output as input. This specification is done by means of trace restrictions.

When we study protocols in isolation, we do not want to consider how the input is created, we only want to consider what properties hold for the input. We model these properties using trace restrictions. While some trace restrictions are related or similar to security properties, trace restrictions do not model protocol security properties, but rather usage of the protocols.

A trace restriction is essentially a predicate on a trace set. We use this predicate as a filter, selecting a subset of the trace set.

Definition 8. Let Π be a protocol set, and let χ be a predicate on $Tr(\Pi)$. Then

$$Tr(\Pi; \chi) = \{\alpha \in Tr(\Pi) \mid \chi(\alpha)\}.$$

In practice, a user first executing a protocol P and then executing a second protocol Q might pass values obtained from P as input to the execution of Q. In a trace, the mechanism for passing the input value is encoded by initializing the parameter for the Q-run of an agent in some role, to a value produced by the P-run of the same agent acting in the same role.

There are two variants to this use. Either the user inputs a value generated by P into one or more executions of the same role of Q, or he inputs a value into just one execution of the Q role. We define two trace restrictions corresponding to these uses, labeled IO and $IO!$.

We say that a protocol P *establishes* an identifier c if for any trace α and every occurrence of events $(inst, end_\ell(r))$, $\ell \in P$, in the trace, $inst$ is defined at c. This condition can be verified by a syntactical analysis.

For the remainder of this section, let Π be a protocol set, let $\Pi' \subseteq \Pi$ be a set where every protocol establishes c, and let $\Pi'' \subseteq \Pi$ be a protocol set where every protocol has d as a parameter in every role. Let $\alpha \in Tr(\Pi)$, and let I contain all subscripts of events in α which correspond to a create event from Π'':

$$I = \{i \mid \alpha_i = (inst_i, create_\ell(r_i)) \wedge \ell \in \Pi''\}.$$

For each $i \in I$, define the sets

$$A_i = \{j \mid \alpha_j = (inst_j, create_\ell(r_j)) \wedge \ell \in \Pi'' \wedge inst_j(d) = inst_i(d)\} \text{ and}$$
$$B_i = \{j \mid \alpha_j = (inst_j, end_\ell(r_i)) \wedge \ell \in \Pi' \wedge inst_j(c) = inst_i(d)\}.$$

The set A_i contains the runs that received the same input as the run that was started in α_i, and B_i contains the set of runs that could have produced this input. We also define the following subsets:

$$A'_i = \{j \in A_i \mid \forall r \in \mathcal{R} : inst_j(r) = inst_i(r)\} \text{ and}$$
$$B'_i = \{j \in B_i \mid \forall r \in \mathcal{R} : inst_j(r) = inst_i(r)\},$$

where the equality is in the sense that either both are defined and equal, or both are undefined. The set A'_i is therefore the subset of A_i of which the runs have the same ρ as α_i, that is, they believe they are communicating with the same partners. The set B'_i has a similar interpretation.

Definition 9. Let $\Pi' \neq \emptyset, \Pi''$ be such that all protocols in $\Pi' \cup \Pi''$ have the same role set. We define the predicates

$$\chi_{IO}(\alpha; \Pi', \Pi'', c, d) \Leftrightarrow \forall i \in I \, \exists j \in B'_i \, \forall k \in A'_i : j < k$$

and

$$\chi_{IO!}(\alpha;\Pi',\Pi'',c,d) \Leftrightarrow \\ \forall i \in I \exists f : A'_i \to B'_i : f \text{ injective} \wedge (\forall j \in A'_i : f(j) < j).$$

The trace restriction $\chi_{IO}(\alpha;\Pi',\Pi'',c,d)$ says that a protocol set Π'' takes its input from a protocol set Π'. As explained before, this means that any run of a role of a protocol in Π'' initializes its input parameter d to a value that has been recorded in c earlier in the trace, in a run of the corresponding role of a protocol in Π'. (The initialization is specified by defining σ only at the input parameters when the run is created.) In the stricter trace restriction, $\chi_{IO!}(\alpha;\Pi',\Pi'',c,d)$ it is required that at most one run of a role of a protocol in Π'' can take as its input a value produced in a run of the corresponding role of a protocol in Π'. In order to ease notation, when $\Pi = \Pi_1 \cup \Pi_2$, Π' is a subset of Π_1 and Π'' is a subset of Π_2, instead of writing $Tr(\Pi;\chi_{IO}(\cdot;\Pi',\Pi'',c,d))$ and $Tr(\Pi;\chi_{IO!}(\cdot;\Pi',\Pi'',c,d))$, we simply write $Tr(\Pi_1\langle c\rangle \cup \Pi_2\langle c/d\rangle)$ and $Tr(\Pi_1\langle c\rangle \cup \Pi_2\langle c!/d\rangle)$ respectively.

Our main goal is to study security properties of component protocols in isolation. When one protocol takes input from another protocol, we do not want to include the second protocol in the analysis. Our strategy is instead to specify preconditions on the input to our protocol, sufficient for the protocol to achieve its goals. We express these preconditions in terms of trace restrictions.

Next, we define what it means for input to be secret. We emphasize that the trace restriction makes no claim about what eventually happens with the input. It may very well not remain secret. But the construction is such that it is *possible* to keep the input secret. As an example, consider the empty protocol with a claim for secrecy of the input identifier. Under a secret trace restriction on the input, the secrecy claim should always be satisfied. To achieve this, we resort to a technical trick. The idea is that since d is a parameter in the run with run identifier rid, the run term $d\sharp rid$ is guaranteed by the semantics not to be known by the adversary or any other execution at the start of the run. Note that $d\sharp rid$ should not be thought of as a locally generated nonce, it is merely a convenient name for something generated elsewhere.

Definition 10. We define the predicate

$$\chi_{secret}(\alpha;\Pi'',d) \Leftrightarrow \forall i \in I : j = \min A_i \wedge inst_j = (rid,\cdot,\cdot) \Rightarrow inst_i(d) = d\sharp rid.$$

A somewhat stronger notion of secrecy is that of session key, where we not only have secrecy, but also a notion of a session.

Definition 11. Let the event α_i have instantiation $inst_i$, label ℓ_i and role r_i,

with $\ell_i \in P_i$. We define the predicate

$$\begin{aligned}\chi_{session}(\alpha;\Pi',d) \Leftrightarrow \\ \forall i \in I \ : (\exists j \in A'_i : inst_j = (rid,\cdot,\cdot) \wedge inst_i(d) = d\sharp rid) \wedge \\ (\exists f : A'_i \to \mathcal{R} : f \text{ injective} \wedge \\ \forall j \in A'_i : \ell_j \in P_i \wedge f(j) = r_j)\end{aligned}$$

Finally, a much simpler concept is that the protocol takes its input from the adversary. In this case, the idea is that the protocol does not really care about where its input comes from, just that it gets its input. We do not believe this is interesting on its own, but it is a useful tool in analysis. One such example is protocols where the chaining nonce is public, for instance NSL variants using signatures instead of public key encryption.

Definition 12. We define the predicate

$$\begin{aligned}\chi_{adversary}(\alpha;\Pi',d) \Leftrightarrow \\ \forall 1 \leq i \leq |\alpha| : \alpha_i = (inst, create_\ell(r)) \wedge \ell \in \Pi' \Rightarrow inst(d) \in M_i^\alpha.\end{aligned}$$

To simplify the notation, we denote these trace sets simply as $Tr(\Pi\langle secret/d\rangle)$, $Tr(\Pi\langle session/d\rangle)$ and $Tr(\Pi\langle adversary/d\rangle)$.

2.5 Security properties

We have already introduced claim events in the trace model. Claim events are not real protocol events, but markers we put in a trace to indicate that a certain statement about the trace is supposed to hold. We can, for instance, extend the role definition of $NSL'(i)$ from Example 3 with claim event $claim_8(i, secret, ni')$, which contains the claim *secret*. If an agent reaches this claim event during his execution of role i, it is intended that an intruder will never learn the value of his nonce.

The definition of security properties proceeds in three steps. First, we define the general signature of a security property, which we consider a predicate on protocol traces. Next, we express that such a property is satisfied if it holds for all traces of the protocol. Finally, we define a number of security properties, such as secrecy and authentication.

In general, a security property is a predicate on the traces of a protocol. So, given protocol Π and a claim cl, $f_{cl}(\Pi, claim_\ell(r, cl, m))$ assigns truth values to pairs $(inst, \alpha)$, where α is a trace of Π and $inst$ is an instantiation of the variables in the claim event.

Definition 13. A *security property* is a function f_{cl}, $cl \in Claim$, that associates with every pair (Π, ev) of a protocol set Π and a role claim event ev with claim cl, a predicate on pairs of instantiations and traces:

$$f_{cl} : \mathcal{P}(Prot) \times \{claim_\ell(r, cl, m) \mid \ell \in \mathcal{L}, r \in \mathcal{R}, m \in RoleTerm\} \rightarrow \bigcup_{\Pi \in \mathcal{P}(Prot)} \{Inst \times Tr(\Pi) \rightarrow \{true, false\}\},$$

where $\mathcal{P}(Prot)$ is the powerset of the set of all protocols.

Note that a context is needed to evaluate an instantiation, and the label ℓ on the role claim event determines this context.

A security property f_{cl} is satisfied in protocol set Π if it yields true for all protocol traces containing this claim cl. This is expressed in the following definition. However, we have included the additional restriction that only claims concerning sessions between trusted agents are evaluated. One cannot expect, for instance, that a shared secret is really secret if one of the communication partners is corrupted. Notice that this does not rule out the possibility that a secret in a trusted session is broken due to an interleaving of a session with untrusted partners. Of course there are security properties for which this restriction is not appropriate, but since the properties used in this paper all have this restriction in common, it is included in the following definition of satisfaction.

Definition 14. Let f_{cl} be a security property, Π a protocol set, and ℓ the label of a claim event with claim cl. We say that Π *satisfies* the claim ℓ, denoted by $\mathrm{sat}(\Pi, \ell)$, if

$$\forall \alpha \in Tr(\Pi) \forall i : \alpha_i = (inst, claim_\ell(r, cl, m)) \Rightarrow f_{cl}(\Pi, claim_\ell(r, cl, m))(inst, \alpha) \vee (inst = (\cdot, \rho, \cdot) \wedge im(\rho) \not\subseteq \mathcal{A}_T).$$

We extend this in the obvious way with trace restrictions and to sets of claim event labels writing $\mathrm{sat}(\Pi, \{\ell_i\}; \chi)$.

As an example, we can claim secrecy for a particular role term m by inserting a suitable claim event into the protocol specification. That claim event will translate into the following statement about a trace α: The adversary never learns the run term $inst(m)$ in the trace α.

Definition 15. Let $\alpha \in Tr(\Pi)$. The security property f_{secret} associates the protocol set Π and the claim event $ev = claim_\ell(r, secret, m)$ with the statement

$$f_{secret}(\Pi, ev)(inst, \alpha) \Leftrightarrow inst(m) \notin M^\alpha_{|\alpha|+1},$$

where the initial intruder knowledge is determined from Π.

We also need to express that a given run is part of a *session* for its protocol. We achieve this by requiring that no two runs of the same role of the protocol have the same value for the *session identifier* (the argument). We call this property *session uniqueness*.

Definition 16. The security property $f_{\textit{session-unique}}$ associates the protocol set Π and the claim event $ev = \textit{claim}_\ell(r, \textit{session-unique}, m)$ with the statement

$$\begin{aligned} f_{\textit{session-unique}}(\Pi, ev)(\textit{inst}, \alpha) \Leftrightarrow \forall 1 \leq i, j \leq |\alpha| : \\ (\alpha_i = (\textit{inst}_i, \textit{claim}_\ell(r, \cdot, \cdot)) \wedge \\ \alpha_j = (\textit{inst}_j, \textit{claim}_\ell(r, \cdot, \cdot)) \wedge \\ \textit{inst}_i(m) = \textit{inst}_j(m) = \textit{inst}(m)) \Rightarrow i = j. \end{aligned}$$

Many protocols establish *session keys*, and we identify three requirements that a session key needs to satisfy:

(1) The session key must be secret.
(2) There must be a session, that is, one run of each role of the protocol must know the session key.
(3) The key must act as a session identifier, that is, it must be unique across all runs of the same role of the same protocol.

The first requirement is taken care of by the secrecy property and the third requirement by the session-unique property. The second requirement is taken care of by data agreement for the session key, which we now define. The idea is that, for every other role in the protocol, there must exist a run that has the same value for the argument term at some event causally preceding the claiming event. These runs must also agree on which agent executes which role, thus possibly forming a session of the protocol.

Definition 17. The security property $f_{\textit{data-agree}}$ associates the protocol set Π and the claim event $ev = \textit{claim}_\ell(r, \textit{data-agree}, m)$, with $\ell \in P$, $P \in \Pi$, with the statement

$$\begin{aligned} f_{\textit{data-agree}}(\Pi, ev)(\textit{inst}, \alpha) \Leftrightarrow \exists \textit{cast} \in \textit{Cast}(P, \alpha) : \\ \forall r' \in \textit{dom}(P)\ \exists j : \alpha_j = (\textit{inst}_j, \cdot) \wedge \\ \textit{inst}_j = (\textit{cast}(r'), \cdot, \cdot) \wedge \textit{inst}_j(m) = \textit{inst}(m). \end{aligned}$$

Note that causal precedence is implicitly required in this definition. Given any trace with a claim event, we can create a new trace by removing any event not causally preceding the claim event. Hence, for the definition to be satisfied, there must be agreeing events for all roles causally preceding the claim event.

Now we can define the session key claim.

Definition 18. The security property $f_{session}$ associates the protocol set Π and the claim event $ev = claim_\ell(r, session, m)$ with the statement

$$\begin{aligned} f_{session}(\Pi, ev)(inst, \alpha) \Leftrightarrow \\ & f_{secret}(\Pi, claim_\ell(r, secret, m))(inst, \alpha) \wedge \\ & f_{session\text{-}unique}(\Pi, claim_\ell(r, session\text{-}unique, m))(inst, \alpha) \wedge \\ & f_{data\text{-}agree}(\Pi, claim_\ell(r, data\text{-}agree, m))(inst, \alpha). \end{aligned}$$

We also define a weaker session key claim, where we drop the requirement about the agreement with (and therefore existence of) communication partners.

Definition 19. The security property $f_{wsession}$ associates the protocol set Π and the claim event $ev = claim_\ell(r, wsession, m)$ with the statement

$$\begin{aligned} f_{wsession}(\Pi, ev)(inst, \alpha) \Leftrightarrow \\ & f_{secret}(\Pi, claim_\ell(r, secret, m))(inst, \alpha) \wedge \\ & f_{session\text{-}unique}(\Pi, claim_\ell(r, session\text{-}unique, m))(inst, \alpha). \end{aligned}$$

Finally, we deal with authentication. Our preferred notion is *synchronization* [16], a strong form of authentication. A non-injective synchronization claim holds if there are executions of the other protocol roles whose *read* and *send* events match the claiming execution's events, up to the claim event. An even stronger notion of authentication, *injective synchronization*, holds if there is exactly one set of executions of the other protocol roles such that the *read* and *send* events match the claiming execution's events, up to the claim event.

Definition 20. The security property f_{synch} associates the protocol set Π and the claim event $ev = claim_\ell(r, synch)$, $\ell \in P$ for some $P \in \Pi$ with the statement

$$\begin{aligned} f_{synch}(\Pi, ev)(inst, \alpha) \Leftrightarrow \exists cast \in Cast(P, \alpha), 1 \leq i \leq |\alpha| : \alpha_i = (inst, ev) \wedge \\ \forall r' \in dom(P) \forall \ell' \in P(r) : \ell' \in P(r') \wedge read_{\ell'} \prec ev \\ \Rightarrow \exists j < k < i : \\ \alpha_j = (inst'', send_{\ell'}(m)) \wedge \alpha_k = (inst''', read_{\ell'}(m')) \wedge \\ inst''(m) = inst'''(m') \wedge \\ ((inst'' = (cast(r), \cdot, \cdot) \wedge inst''' = (cast(r'), \cdot, \cdot)) \vee \\ (inst'' = (cast(r'), \cdot, \cdot) \wedge inst''' = (cast(r), \cdot, \cdot))). \end{aligned}$$

Definition 21. The security property $f_{i\text{-}synch}$ associates the protocol set Π and the claim event $ev = claim_\ell(r, i\text{-}synch)$, $\ell \in P$ for some $P \in \Pi$ with the

statement

$$
\begin{aligned}
f_{i\text{-}synch}(\Pi, ev)(inst, \alpha) \Leftrightarrow \exists! cast \in Cast(P, \alpha), 1 \leq i \leq |\alpha| : \alpha_i = (inst, ev) \wedge \\
\forall r' \in dom(P) \forall \ell' \in P(r) : \ell' \in P(r') \wedge read_{\ell'} \prec ev \\
\Rightarrow \exists j < k < i : \\
\alpha_j = (inst'', send_{\ell'}(m)) \wedge \alpha_k = (inst''', read_{\ell'}(m')) \wedge \\
inst''(m) = inst'''(m') \wedge \\
((inst'' = (cast(r), \cdot, \cdot) \wedge inst''' = (cast(r'), \cdot, \cdot)) \vee \\
(inst'' = (cast(r'), \cdot, \cdot) \wedge inst''' = (cast(r), \cdot, \cdot))).
\end{aligned}
$$

Certain security properties can be evaluated by merely looking at the events in a trace that belong to the protocol in which the claim was made. This class of properties are called protocol-centric, and as we will see, we can prove theorems that apply to all properties in this class. Since authentication properties are concerned with the occurrence of events of the given protocol, they are typical members of this class.

Definition 22. Let P be a protocol, and Π a protocol set. Denote by π_P and π_Π the maps on traces that remove any protocol event that does not belong to P or a protocol in Π, respectively.

Let s be any bijection on the set of nonce run terms $\{c\sharp rid \mid c \in \mathcal{ID}, rid \in Runid\} \cup \mathcal{IT}$. This is basically a renaming of nonce run terms. Any such bijection can be naturally extended to a bijection on the set of run terms. For any trace α, we define $s(\alpha)$ to be the trace where every instantiation $inst$ is replaced with $s \circ inst$.

Definition 23. We say that a security property f_{cl} is *protocol-centric* if for any (Π, ev) such that f_{cl} is defined and ev belongs to a protocol $P \in \Pi$, and for any renaming s on nonce run terms,

$$
\begin{aligned}
\forall \alpha, \alpha' \forall inst \in Inst : s(\pi_P(\alpha)) = \pi_P(\alpha') \\
\Rightarrow f_{cl}(\Pi, ev)(inst, \alpha) = f_{cl}(\Pi, ev)(s \circ inst, \alpha').
\end{aligned}
$$

(The renaming s is included in this definition for technical reasons.)

We observe that *session-unique*, *data-agree*, *synch* and *i-synch* are protocol-centric, while *secret* and therefore *session* are not.

3 Framework for reasoning

Automatically proving large protocols secure is computationally challenging. If the protocol can be split into a sequential composition of several subprotocols, one approach to verification is to analyze the subprotocols in a common context. Properties proved for each of the subprotocols running in this multi-protocol context can often be used to deduce properties for the composed protocol. Note that some authors consider protocols running in a multi-protocol context to be composed in parallel. We use "composition" only about operations that combine two or more protocol objects into a new protocol object. The only form of composition considered in this paper is sequential composition.

Unfortunately, automatically verifying protocol properties in such a multi-protocol context is not computationally easier than analysing the composed protocol itself. The ideal is to study each subprotocol in perfect isolation, without consideration of any other protocols. Our approach is to study under which conditions protocols running in parallel can be shown not to interfere with each other, such that results obtained by analysis in isolation will be valid for a multi-protocol context.

Our approach is to use a very strong, but efficiently verifiable notion of independence between protocols. We show how to design protocols to ensure such independence without incurring any significant performance penalty. We then prove a number of theorems showing conditions under which protocols can run in a multi-protocol context without interfering with each other. Finally, we define sequential protocol composition (similar to the one in PCL) and show how properties of subprotocols can be combined to give security properties for the composed protocols.

We use the formal model described in Section 2, with two restrictions:

(1) We require that the partial function σ in instantiations assigns only nonce run terms to variables and parameters.
(2) We restrict ourselves to protocols that use secret long-term keys only as keys, never as content of messages.

The first restriction is essential for the notion of strong independence, defined in the next section. We refer to Section 6 for a discussion of the implications. The second restriction could be lifted, but this would subtly complicate analysis, since the use of long-term secret keys becomes much harder to predict.

We say that two protocols are independent if no encryption term produced by the first protocol running in the context of the second protocol will be decrypted or verified by the second protocol, and vice versa. Formally:

Definition 24. Let Π_1, Π_2 be two disjoint protocol sets, and let χ be a (possibly empty) trace restriction. We say that Π_1 *and* Π_2 *are independent in the context of* χ, denoted $\text{indep}(\Pi_1, \Pi_2; \chi)$ (alternatively if χ is empty, Π_1 *and* Π_2 *are independent*, denoted $\text{indep}(\Pi_1, \Pi_2)$), if

$$\begin{aligned}
&\forall \alpha \in \mathit{Tr}(\Pi_1 \cup \Pi_2; \chi)\ \forall x, y, x', y' \in \mathit{RoleTerm}: \\
&\Big(\alpha_i = (\mathit{inst}, \mathit{send}_\ell(m)) \wedge \Big(\alpha_j = (\mathit{inst}', \mathit{read}_{\ell'}(m')) \vee \alpha_j = (\mathit{inst}', \mathit{send}_{\ell'}(m')) \vee \\
&\qquad\qquad \alpha_j = (\mathit{inst}', \mathit{claim}_{\ell'}(\cdot, \cdot, m'))\Big) \wedge \\
&\quad \Big(\{\!|\, x \,|\!\}_y \sqsubseteq m \wedge \{\!|\, x' \,|\!\}_{y'} \sqsubseteq m' \wedge \mathit{inst}(\{\!|\, x \,|\!\}_y) = \mathit{inst}'(\{\!|\, x' \,|\!\}_{y'})\Big)\Big) \\
&\Rightarrow (\ell, \ell' \in \Pi_1 \vee \ell, \ell' \in \Pi_2).
\end{aligned}$$

In general, proving independence is a non-trivial problem, but for many protocol sets it is easy in the sense that the protocol sets satisfy an even stronger notion of independence. We say that two protocol sets are strongly independent if they have no encryptions of the same form. Unlike independence, strong independence can be easily verified at the syntactical level, and it implies independence. Note that different protocols can use the same cryptographic keys and still be strongly independent, and thus independent.

Definition 25. Let Π_0 and Π_1 be two disjoint protocol sets. We say that Π_0 *and* Π_1 *are strongly independent*, denoted $\text{s-indep}(\Pi_0, \Pi_1)$, if for any $b \in \{0, 1\}$, any role specification $(\mathit{elist} \cdot \mathit{send}(m) \cdot \mathit{elist}', \mathit{type})$ in a protocol in Π_b, any role terms x, y, any role specifications $(\mathit{elist}'' \cdot \mathit{send}(m') \cdot \mathit{elist}''', \mathit{type}')$, $(\mathit{elist}'' \cdot \mathit{read}(m') \cdot \mathit{elist}''', \mathit{type}')$ or $(\mathit{elist}'' \cdot \mathit{claim}(r, c, m') \cdot \mathit{elist}''', \mathit{type}')$ in protocols of Π_{1-b}, any map s on the set $\mathcal{ID}$ and any map s' on the set $\mathcal{R}$,

$$\{\!|\, x \,|\!\}_y \sqsubseteq m \Rightarrow \{\!|\, s(s'(x)) \,|\!\}_{s(s'(y))} \not\sqsubseteq m'.$$

Note that any map s on identifiers and s' on roles naturally induce maps on the set of role terms and we identify these maps with s and s'. Also note that since strong independence is a syntactical property, there is no need to consider traces or trace restrictions.

Theorem 26. *If two protocol sets* Π_1 *and* Π_2 *are strongly independent, then they are independent.*

Proof. Obvious from the fact that only nonce run terms are assigned to variables and parameters. □

The notion of strongly independent protocols is obviously very strong, and there are many independent protocols that are not strongly independent. However, it is possible to verify strong independence by a simple syntactical check on the protocols. Obviously, strong independence may not be useful for analysing some existing protocols, but it does cover many deployed protocols (see Section 4).

One way to achieve strong independence is to use separate key infrastructures for every protocol. Unfortunately, this is expensive and wasteful. A more practical way to get strong independence is through *protocol tags*. Every protocol is given a unique tag which is embedded into every ciphertext the protocol makes. This trivially implies strong independence.

Protocol tags may be a desirable approach for protocol design, since they typically require no extra bandwidth, and only modest extra computational effort.

Modern signature schemes typically process the message to be signed with a hash function, create a signature tag, and attach the tag to the message. Adding a protocol tag of reasonable length (say 128 bits) to the message will usually result in a minor increase in the cost of computing the hash function. Since the signature is simply attached to the message, and both signer and verifier know the protocol tag, there is no need to actually transmit the protocol tag. The signer can remove the tag from the message before transmitting, the verifier puts the tag back in before verifying. A signature made by one protocol will not pass the verification by a second protocol.

Modern encryption schemes typically allow for part of the message to be left unencrypted but authenticated. Again, if we include the protocol tag in the unencrypted part, the encrypter can remove the tag before transmission and the decrypter can insert the tag prior to decryption. Typically, a protocol tag of reasonable length will result in a minor increase in the cost of authentication.

As for hash-like function evaluations, most cryptographically interesting functions either allow the protocol tag to be inserted into the function evaluation, or allow cryptographic separation by choosing distinct parameters. The computational cost of most such measures are expected to be modest.

To summarize, protocol tags typically have no bandwidth cost and modest computational cost. This suggests that protocol tagging is a viable and sensible strategy for protocol design.

Once we have independence, we are ready to prove that any protocol remains correct in the presence of independent protocols. The general idea for proving all of the results in this section is to define maps between trace sets, and then argue that the predicate derived from a claim statement remains unchanged under this map.

The next theorem says that if one protocol set keeps something secret, it will keep it secret even in the presence of a second, but independent, protocol set.

Theorem 27. *Let* Π_1 *and* Π_2 *be two independent protocol sets. Let* ℓ *be the label of some secret claim event in* Π_1*. Then*

$$\mathrm{sat}(\Pi_1, \ell) \Rightarrow \mathrm{sat}(\Pi_1 \cup \Pi_2, \ell).$$

Proof. Let S be the set of nonce run terms in α originating in runs of roles from protocols in Π_2. Let $s : S \to \mathcal{IT}$ be an injection such that no term in the image of s appears in α. We can extend s to the set of nonce run terms by letting s be the identity where it is not already defined. This map can then be extended naturally to a renaming map on the set of run terms.

We construct a new trace α' from α by removing any events belonging to runs of roles from protocols in Π_2, and replacing any other event $(inst, ev)$ by $(s \circ inst, ev)$. (Since s renames only nonce run terms, the composition $s \circ inst$ may affect only the σ function of $inst$.) Note that by independence, if any nonce run term originates in a run of a role of a protocol in Π_2, the only way a run of a role of a protocol in Π_1 will read that nonce run term is if the adversary also knows that nonce run term. From the semantics, we have that a nonce of Π_2 can only occur in a run of a role of Π_1 as a subterm of an instantiated variable. Therefore, if we replace the nonce run term by an attacker-generated nonce run term (such that the type constraints on the containing variable are met), the trace will still be valid even after the Π_2-events are removed. This means that $\alpha' \in Tr(\Pi_1)$.

There is always a canonical choice of injection s (given the well-ordering on the nonce run terms induced by the natural numbers), and this gives us a map

$$\tau : Tr(\Pi_1 \cup \Pi_2) \to Tr(\Pi_1). \tag{1}$$

Note that $\tau = s \circ \pi_{\Pi_1}$.

Now consider a run term t claimed secret in α. In α', the corresponding run term is $s(t)$, and we know that this is secret. We first determine why $s(t)$ is secret, and we may as well assume that $s(t)$ is a non-tuple run term. If $s(t)$

has the form $f(u)$ for some function f and run term u, then t is secret by assumption. If $s(t)$ is a nonce run term, we know that first of all $s(t)$ must originate in a run of a role of a protocol in Π_1. Second, every time it appears in a sent run term, it must be inside an encryption term. By independence, no Π_2-run will decrypt that ciphertext. Therefore, t must be secret in α. Otherwise, t must have the form $\{\!|\, u \,|\!\}_v$ for some run terms u and v. If $s(u)$ is secret, we must show that u is secret. We consider u instead of t and return to the start of the argument. Otherwise, $s(v)$ must be secret. By independence, it is sufficient to show that v is secret, so we consider v instead of t and return to the start of the argument.

Since terms cannot be infinitely nested, this argument chain must eventually stop, and in the process prove that t is secret in α. This concludes the proof. □

If secrecy of some nonce is not important for satisfying some secrecy claim in some protocol set, then passing the nonce to an independent protocol set will not compromise the secrecy claim. The intuition is that the worst an independent protocol can do is to reveal the nonce to the intruder, and therefore we only need to analyse what happens in that case.

Definition 28. Let P be a protocol establishing c. Then $P\langle c^* \rangle$ is the protocol

$$P\langle c^* \rangle = \{r \mapsto s \cdot \mathit{send}(r, \nu_{r_0,r_1}(r), c) \cdot \mathit{end}(r) \mid P(r) = s \cdot \mathit{end}(r)\},$$

where r_0 and r_1 are two distinct roles of P and $\nu_{r_0,r_1}(r)$ is r_0 when $r \neq r_0$, otherwise r_1.

We extend this notation to protocol sets in the obvious way, writing $\Pi\langle c^* \rangle$.

Theorem 29. *Let $\Pi_1\langle c^* \rangle$ and $\Pi_2\langle \mathit{adversary}/d \rangle$ be two independent protocol sets and let ℓ be the label of some secret claim event in Π_1. Then*

$$\mathrm{sat}(\Pi_1\langle c^* \rangle, \ell) \Rightarrow \mathrm{sat}(\Pi_1\langle c \rangle \cup \Pi_2\langle c/d \rangle, \ell).$$

Proof. It is clear that $\mathit{Tr}(\Pi_1\langle c \rangle \cup \Pi_2\langle c/d \rangle)$ embeds naturally in $\mathit{Tr}(\Pi_1\langle c^* \rangle \cup \Pi_2\langle \mathit{adversary}/d \rangle)$. Furthermore, under this embedding the intruder knowledge is strictly increased. By Theorem 27, the secrecy claim holds in the latter trace set. It must therefore also hold in the former trace set and the theorem is proven. □

If secrecy of some nonce may be important for some secrecy claim, then passing the nonce to an independent protocol set that preserves the secrecy of its input will not compromise the secrecy claim. (Note that the secrecy claims in the second protocol must be positioned at the start of the role. Otherwise, the

protocol would be allowed to compromise the secrecy of the input as long as none of its roles reaches its secrecy claim.)

Theorem 30. *Let $\Pi_1\langle c\rangle$ and $\Pi_2\langle c/d\rangle$ be two protocol sets, let $\Pi_1' \subseteq \Pi_1$ be a set of protocols establishing c and $\Pi_2' \subseteq \Pi_2$ be a set of protocols with d as a parameter. Let $\ell \in \Pi_1 \cup \Pi_2$ be the label of some secret claim event. If Π_1 and Π_2 are independent under the trace restrictions $\chi_{IO}(\cdot; \Pi_1', \Pi_2', c, d)$ and $\chi_{secret}(\cdot; \Pi_2', d)$, then*

$$\mathrm{sat}(\Pi_1, \{\ell_i\} \cup \{\ell\}) \wedge \mathrm{sat}(\Pi_2\langle secret/d\rangle, \{\ell_i'\} \cup \{\ell\}) \Rightarrow \mathrm{sat}(\Pi_1\langle c\rangle \cup \Pi_2\langle c/d\rangle, \ell).$$

Here we assume that $\{\ell_i\}$ are the labels of claim events $\{claim_{\ell_i}(r_i, secret, c)\}$ in every role of every protocol in Π_1 that establishes c and the $\{\ell_i'\}$ are labels of claim events $\{claim_{\ell_i'}(r_i', secret, d)\}$ in every role of every protocol in Π_2 that has d as a parameter. The event $claim_{\ell_i'}(r_i', secret, d)$ is assumed to occur before any send or read event in the role specification.

Proof. Let $\alpha \in Tr(\Pi_1\langle c\rangle \cup \Pi_2\langle c/d\rangle)$ be a trace. Let I be the set of indexes such that $\alpha_i = (inst_i, create(r))$ for some role r of a protocol Q in Π_2 that takes c as input for the parameter d. For each i, define the set

$$A_i = \{j \in I \mid inst_j(d) = inst_i(d)\}.$$

Let rid_i be such that $\alpha_{\min A_i} = ((rid_i, \cdot, \cdot), \cdot)$. Note that for any $i \in I$, the nonce run term $d\sharp rid_i$ never appears in α. Let $S = \{inst_i(d) \mid i \in I\}$ and $S' = \{d\sharp rid_i \mid i \in I\}$, and let s be the substitution that maps $inst_i(d)$ to $d\sharp rid_i$ for all i in I.

We construct a new trace α' from α by replacing any event $(inst, ev)$ that belongs to Π_2 by the event $(s \circ inst, ev)$.

We claim that $\alpha' \in Tr(\Pi_1 \cup \Pi_2\langle secret/d\rangle)$, and we get a map

$$\tau : Tr(\Pi_1\langle c\rangle \cup \Pi_2\langle c/d\rangle) \to Tr(\Pi_1 \cup \Pi_2\langle secret/d\rangle), \tag{2}$$

along with a natural bijection

$$\theta : M^{\alpha} \to M^{\alpha'}.$$

We will prove the claim and construct the map θ by induction. The theorem will then follow from a simple observation.

Suppose $\alpha_0' \cdots \alpha_{j-1}'$ is a valid trace. The subterm relation $\sqsubseteq$ defines a partial ordering on M_j^{α} and $M_j^{\alpha'}$. Let U_j and U_j' be the minimal elements of these

sets, together with the terms that cannot be inferred from smaller terms:

$$U_j = \{x \in M_j^\alpha \mid t \sqsubseteq\text{-minimal}\} \cup \{\{\!|\, t \,|\!\}_{t'} \in M_j^\alpha \mid t \notin M_j^\alpha \vee t' \notin M_j^\alpha\},$$
$$U'_j = \{x \in M_j^{\alpha'} \mid t \sqsubseteq\text{-minimal}\} \cup \{\{\!|\, t \,|\!\}_{t'} \in M_j^{\alpha'} \mid t \notin M_j^{\alpha'} \vee t' \notin M_j^{\alpha'}\}.$$

Note that $M_j^\alpha = \overline{U_j}$ and $M_j^{\alpha'} = \overline{U'_j}$. Also note that any encryption term in U_j and U'_j must originate from some *send* event.

Define the sets

$$V_j = \{t \in U_j \mid \exists t' \in S : t' \sqsubseteq t\} \text{ and } V'_j = \{t \in U'_j \mid \exists t' \in S \cup s(S) : t' \sqsubseteq t\}.$$

Let $V_{j,1}$ and $V_{j,2}$ be the subsets of V_j of elements originating in Π_1 and Π_2, respectively. Let

$$V'_{j,1} = \{t \in U'_j \mid \exists t' \in S : t' \sqsubseteq t\}$$
$$V'_{j,2} = \{t \in U'_j \mid \exists t' \in s(S) : t' \sqsubseteq t\}$$

Let $W_j = U_j \setminus V_j$ and $W'_j = U'_j \setminus V'_j$.

We can now prove the claim and construct the map θ by induction on j. The induction hypothesis is that $\alpha'_0 \cdots \alpha'_{j-1}$ is a valid trace and the following two properties hold for the structure of the intruder knowledge:

(1) $S \cap V_j = \emptyset$ and $(S \cup s(S)) \cap V'_j = \emptyset$.
(2) There exists a bijection $\theta : U_j \to U'_j$ such that θ restricted to $W_j \cup V_{j,1}$ is the identity map, and θ restricted to $V_{j,2}$ corresponds to the map induced by the substitution s.

Note that the bijection θ extends to a bijection $\theta : M_j^\alpha \to M_j^{\alpha'}$.

The induction basis is trivially satisfied for the empty trace and easy to verify for α'_0.

The trace restriction on d is satisfied by design, so if α_j is a *create* event, $\alpha'_0 \cdots \alpha'_j$ is a valid trace. Also, the same substitution is applied to all events in a run, so if α_j is a *send*, *claim* or *end* event, $\alpha'_0 \cdots \alpha'_j$ is a valid trace, because the instantiations will be consistent with the run (basically the instantiation of the last event).

Next, we consider a *read* event $\alpha_j = (inst, read_\ell(m))$, $\alpha'_j = (inst', read_\ell(m))$. We must prove that $inst'(m) \in M_j^{\alpha'}$. We know that $inst(m) \in M_j^\alpha$. If $inst(m) \in \overline{W_j}$ we are done, and $\alpha'_0 \cdots \alpha'_j$ is a valid trace.

By independence we know that $V_{j,1} \cap V_{j,2} = \emptyset$. Again by independence, if $\ell \in \Pi_b$, then any run term in V_j that is a subterm of $inst(m)$ is also in $V_{j,b}$, and

we get that $inst(m)$ is in the closure of $V_{j,b} \cup W_j$. Note that $V'_{j,1} \cup W'_j = V_{j,1} \cup W_j$ and $V'_{j,2} \cup W'_j = s(V_{j,2} \cup W_j)$. If $\ell \in \Pi_1$ we have that $inst'(m) = inst(m) \in M_j^{\alpha'}$. If $\ell \in \Pi_2$ we must have that $inst'(m) = s(inst(m)) \in M_j^{\alpha'}$. Therefore, under the induction hypothesis, $\alpha'_0 \cdots \alpha'_j$ is a valid trace.

We finish the inductive step by showing that (1) and (2) are also satisfied after the jth event. We need only consider the event $\alpha_j = (inst, send_\ell(m))$, $\alpha'_j = (inst', send_\ell(m))$. The only interesting inference rule is $(\{\!|\, t \,|\!\}_{t'}, t') \Rightarrow t$, and we will show that the structure is unchanged by decryptions, up to some trivial rewriting.

For any set of run terms T, let $rcl(T)$ be the smallest set of run terms containing T that is closed under tuple creation and dissolution, encryption with known keys and removing signatures. This is the restricted closure, closure without decryptions. Note that $M_j^\alpha = rcl(U_j)$.

Under the induction hypothesis for U and U', if we augment U by decrypting a run term $\{\!|\, t \,|\!\}_{t'}$, where $t' \in rcl(U)$, then we can augment U' by decrypting the run term $\theta(\{\!|\, t \,|\!\}_{t'})$, since $\theta(t') \in rcl(U')$. If $\theta(t) = t$, then clearly we augment U and U' in the same way. Likewise, if $\theta(t) \neq t$, then $\theta(t)$ and t are equal up to substitution by s, and U and U' are augmented in the same way, up to substitution. This means that $S \cap U = \emptyset$, because we know that $(S \cup s(S)) \cap U' = \emptyset$. The maps can therefore be extended, and (1) and (2) still hold true after augmentation. Finally, if some of the elements in U are no longer minimal and are not encryptions that should be preserved, then the corresponding elements in U' will no longer be minimal, nor be encryptions that should be preserved. The other direction also holds. Therefore, we can discard all superfluous elements.

The list U_{j+1} can be reached from U_j by adding the run terms obtained from the send event to the list, then performing a finite sequence of decryption operations, then possibly discarding some elements from the list. The above argument shows that the same operations (up to substitution) will turn U'_j into U'_{j+1} in such a way that (1) and (2) still hold for U_{j+1} and U'_{j+1}. This completes the inductive step.

To complete the proof of the theorem, we first observe that for any secrecy claim event $\alpha_j = (inst, claim_\ell(\cdot, \cdot, m))$ there is $\alpha'_j = (inst', claim_\ell(\cdot, \cdot, m))$ and note that $\mathrm{sat}(\Pi_1 \cup \Pi_2\langle secret/d\rangle, \ell)$ is true by Theorem 27, thus $inst'(m) \notin M^{\alpha'}$. Next, if $inst(m) \in M^\alpha$, then $\theta(inst(m))$ would be defined and equal to $inst'(m)$, contradicting $inst'(m) \notin M^{\alpha'}$. We conclude that $inst(m) \notin M^\alpha$ and the secrecy claim holds. □

The following theorem states that protocol-centric claims remain valid if we execute a protocol in the context of another protocol that is independent of

the first.

Theorem 31. *Let Π_1 and Π_2 be two independent protocol sets, and let ℓ be the label of some claim event in Π_1. If the security property associated with ℓ is protocol-centric, then*

$$\mathrm{sat}(\Pi_1, \ell) \Rightarrow \mathrm{sat}(\Pi_1 \cup \Pi_2, \ell).$$

Proof. Since Π_1 and Π_2 are independent, we can use the trace map τ from (1) on page 25. Let P be the protocol where the claim event with label ℓ appears. By construction of the τ map, we have $s(\pi_P(\alpha)) = \pi_P(\tau(\alpha))$, for some substitution s. Since the claim is protocol-centric, we are done. □

When one protocol establishes session for some nonce run term, and a second protocol expects a session key as input, we can use the nonce run term from the first protocol as input to the second, without compromising any protocol-centric security properties.

Note that in the following two results, we restrict to exactly one protocol creating output and one protocol taking input. This is for simplicity, and can easily be solved using protocol tags to create many distinct variants of a single protocol.

Theorem 32. *Let Π_1 and Π_2 be two protocol sets, such that $P \in \Pi_1$ is the only protocol establishing c and $Q \in \Pi_2$ is the only protocol taking c as input for d. Let ℓ be the label of a protocol-centric claim event in Π_1 or Π_2. If Π_1 and Π_2 are independent under the trace restrictions $\chi_{IO!}(\cdot; P, Q, c, d)$ and $\chi_{secret}(\cdot; Q, d)$, then*

$$\mathrm{sat}(\Pi_1, \{\ell_i\} \cup \{\ell\}) \wedge \mathrm{sat}(\Pi_2\langle session/d\rangle, \{\ell'_i\} \cup \{\ell\}) \Rightarrow \mathrm{sat}(\Pi_1\langle c\rangle \cup \Pi_2\langle c!/d\rangle, \ell),$$

where $\{\ell_i\}$ are labels of claim events $\{claim_{\ell_i}(r_i, session, c)\}$ in every role of P, and $\{\ell'_i\}$ are labels of claim events $\{claim_{\ell'_i}(r'_i, secret, d)\}$ in every role Q, occurring before any send or read event in the role specification.

Proof. Let ℓ be in a protocol R. First, we note that the map $\tau : Tr(\Pi_1\langle c\rangle \cup \Pi_2\langle c!/d\rangle) \rightarrow Tr(\Pi_1 \cup \Pi_2\langle secret/d\rangle)$ from (2) on page 27 exists, since the *session* claims imply corresponding *secret* claims. Since for some substitution s, $s(\pi_R(\alpha)) = \pi_R(\tau(\alpha))$, we only need to show that the *session* trace restriction is satisfied for the input to Q, and the result will follow from Theorem 31. Because of the *data-agree* claims, we have a full session for P, and by the *session-unique* claims, this session is unique. Thus for any *end* event for any role r of P, there are no other *end* events for that role with the same value for c. Hence, the *session* trace restriction is satisfied. □

Finally, we combine the previous theorems into a statement about preservation of session secrecy.

Corollary 33. *Let Π_1 and Π_2 be two protocol sets, such that $P \in \Pi_1$ is the only protocol establishing c and $Q \in \Pi_2$ is the only protocol taking c as input for d. Let ℓ be the label of a (weak) session claim in Π_1 or Π_2. If Π_1 and Π_2 are independent under the trace restrictions $\chi_{IO!}(\cdot; P, Q, c, d)$ and $\chi_{secret}(\cdot; Q, d)$, then*

$$\mathrm{sat}(\Pi_1, \{\ell_i\} \cup \{\ell\}) \wedge \mathrm{sat}(\Pi_2\langle session/d\rangle, \{\ell'_i\} \cup \{\ell\}) \Rightarrow \mathrm{sat}(\Pi_1\langle c\rangle \cup \Pi_2\langle c!/d\rangle, \ell),$$

where $\{\ell_i\}$ are labels of claim events $\{claim_{\ell_i}(r_i, session, c)\}$ in every role of P, and $\{\ell'_i\}$ are labels of claim events $\{claim_{\ell'_i}(r'_i, secret, d)\}$ in every role Q, occurring before any send or read event in the role specification.

Proof. First we apply Theorem 27 and Theorem 31 to establish $\mathrm{sat}(\Pi_1 \cup \Pi_2\langle session/d\rangle, \{\ell_i\} \cup \{\ell'_i\})$ (separately establishing the three parts of the session claims: secret, session-unique and data-agree). The results follows by further applications of Theorem 30 and Theorem 32. □

3.3 Composition

In this section we study sequential composition and show how certain security properties of a composed protocol follow from security properties of the subprotocols analyzed in a multi-protocol setting.

As discussed in Section 1, sequential composition (without passing information) of two protocols does not in general preserve synchronisation. The problem is that if there is no mechanism to bind the two subprotocols to each other in the composed protocol, different runs can be interleaved with each other, breaking synchronisation. Therefore, there must be a mechanism that connects the two subprotocols. We achieve this by letting the first subprotocol pass information to the next subprotocol. (A slightly more general definition of chaining composition allowing more than one parameter appears in PCL.)

Definition 34. Let $rs_1 = (elist_1 \cdot end(r), type_1)$ and $rs_2 = (create(r) \cdot elist_2, type_2) \in RoleSpec$. The *sequential composition of role specifications* rs_1 and rs_2 is the role specification $rs_1 \cdot rs_2 = (elist_1 \cdot elist_2, type)$ where

$$type(x) = \begin{cases} type_1(x) & \text{if } type_1(x) \text{ is defined;} \\ type_2(x) & \text{if } type_1(x) \text{ is undefined and } type_2(x) \text{ is defined;} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Definition 35. Let P and Q be two protocols such that $dom(P) = dom(Q)$, $\mathcal{ID}(P) \cap \mathcal{ID}(Q) = \emptyset$. If P establishes c, and d is a parameter in all roles of Q, the *chaining composition* $P \cdot Q$ of P and Q is defined as:

$$P \cdot Q \stackrel{\text{def}}{=} \Big\{ r \mapsto P(r) \cdot Q(r)[c/d] \;\Big|\; r \in dom(P) \Big\},$$

where $Q(r)[c/d]$ denotes replacing d by c in the role specification $Q(r)$.

Note that every event is relabeled after this composition, but there is a natural correspondence between the labels of P and Q, and the labels of $P \cdot Q$ (excluding the *end* event of P and *create* event of Q).

The formal model described in Section 2 allows us to define explicitly the concept of passing information from one protocol to another. This exactly coincides with the idea of input and output, modeled using parameters. However, simply passing information does not suffice to preserve synchronization. Intuitively, if agents of the first subprotocol do not all share the value to be passed on to the next subprotocol, a mismatch between different runs of the agents may occur and synchronization may be broken. Likewise, the next subprotocol must ensure that all agents got passed the same value. In order to ensure this, we use data agreement for the value passed between subprotocols.

Theorem 36. *Let P, Q be protocols such that P establishes c, d is a parameter in all roles of Q and $P \cdot Q$ is defined. Let Π be a set of protocols, $P, Q \notin \Pi$. Let $\{\ell_i\}$ be a set of labels for one injective synchronization claim event and one data agreement claim event with argument c in every role of P, the synchronization claim events appearing after all read and send events in the role. Let ℓ and ℓ' be the labels of a data agreement claim event with d as argument and a synchronization claim event, respectively, in some role Q such that the claim event labelled ℓ causally precedes the one labelled ℓ'. Let ℓ'' be the label of a corresponding injective synchronization claim event in $P \cdot Q$. Then*

$$\mathrm{sat}(\{P\langle c\rangle, Q\langle c!/d\rangle\} \cup \Pi, \{\ell_i\} \cup \{\ell, \ell'\}) \Rightarrow \mathrm{sat}(\{P \cdot Q\} \cup \Pi, \ell'').$$

Proof. Let $\alpha \in Tr(\{P \cdot Q\} \cup \Pi)$. We map α to a trace $\alpha' \in Tr(\{P\langle c\rangle, Q\langle c!/d\rangle\} \cup \Pi)$ as follows: Without loss of generality we can assume that the set *Runid* of run identifiers is a subset of non-negative integers. Let the highest occurring run identifier in α be rid_B, and suppose we have a run with run identifier rid of a role r of $P \cdot Q$ with events $\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_k}$. If there is no event in the run corresponding to an event of the protocol Q, we relabel all of the events to be events of P. Otherwise, let i_l be the first event in the run corresponding to an event of Q.

(1) We relabel $\alpha_{i_1}, \ldots, \alpha_{i_{l-1}}$ to be events of P, and $\alpha_{i_l}, \ldots, \alpha_{i_k}$ to be events of Q.

(2) We change the σ-parts of the instantiations of $\alpha_{i_l}, \ldots, \alpha_{i_k}$ so that they are only defined at identifiers of Q.
(3) We change the run identifier of $\alpha_{i_l}, \ldots, \alpha_{i_k}$ to $rid + rid_B + 1$.
(4) We apply to every other event in the trace the substitution $\{x\sharp rid \mapsto x\sharp rid + rid_B + 1 \mid x \in \mathcal{ID}(Q) \wedge type(x) = const\}$, where $Q(r) = (elist, type)$.
(5) We insert a *end* for $P(r)$ and a create event $Q(r)$ with the proper value for the parameter d just before α_{i_l} in the trace.

When this operation is performed for every run of a role of $P \cdot Q$, we get a trace $\alpha' \in Tr(\{P\langle c\rangle, Q\langle c!/d\rangle\} \cup \Pi)$, and this gives us a map

$$\tau : Tr(\{P \cdot Q\} \cup \Pi) \to Tr(\{P\langle c\rangle, Q\langle c!/d\rangle\} \cup \Pi). \tag{3}$$

Now consider a run of role r with run identifier rid_r where the claim event with label ℓ'' occurs. First, we note that because of injective synchronization for P, we have a unique cast *cast* for P in α'. This translates into a potential cast *cast*$'$ for Q in α' given by $cast'(r') = cast(r) + rid_B + 1$, as well as a cast for $P \cdot Q$ in α. By data agreement for c in P, we know that every run in the cast agree on the value of c. Since P does not take any input, the value of c must originate with one of the roles, hence it must also be unique among all the runs of P. Further, by data agreement on d in Q, we know that *cast*$'$ really is a cast for Q in α', it is unique, that every member of the cast agrees on the value of d, and that this value is the same as the value of c.

Now we verify the *i-synch* claim with label ℓ'' for rid_r in α with the unique cast *cast*. Consider a role r' and a label ℓ_x such that $\ell_x \in P{\cdot}Q(r)$ and $\ell_x \in P{\cdot}Q(r')$. We must show that there are two events with this label in α, belonging to the cast, and sending and reading the same message. Because of synchronization in α', we find matching events belonging to the cast for the corresponding labels in α'. Note that since the map τ only changes instantiations by applying a substitution, if the content of the messages is the same in α', the same must hold in α. We conclude that the injective synchronization claim with label ℓ'' is satisfied in α. □

The following theorem states the conditions under which secrecy is preserved in a sequential protocol composition.

Theorem 37. *Let P, Q be protocols such that P establishes c, d is a parameter in all roles of Q and $P \cdot Q$ is defined. Let Π be a set of protocols, $P, Q \notin \Pi$. Let ℓ be the label of a secret claim event in P or Q, and let ℓ' be the corresponding label in $P \cdot Q$. Then*

$$\mathrm{sat}(\{P\langle c\rangle, Q\langle c!/d\rangle\} \cup \Pi, \ell) \Rightarrow \mathrm{sat}(\{P \cdot Q\} \cup \Pi, \ell').$$

Proof. We use the map τ from (3). Note that instantiations are changed by at most a nonce run term renaming under this map, so the intruder's knowledge is also changed by at most a nonce run term renaming. The predicate derived from the secret claim does not change its value under nonce run term renaming, from which the result follows. □

The same conditions that preserve secrecy, also preserve *session-unique* and *data-agree* in a sequential protocol composition.

Theorem 38. *Let P, Q be protocols such that P establishes c, d is a parameter in all roles of Q and $P \cdot Q$ is defined. Let Π be a set of protocols, $P, Q \notin \Pi$. Let ℓ be the label of a session-unique or data-agree claim event in P or Q, and let ℓ' be the corresponding label in $P \cdot Q$. Then*

$$\mathrm{sat}(\{P\langle c\rangle, Q\langle c!/d\rangle\} \cup \Pi, \ell) \Rightarrow \mathrm{sat}(\{P \cdot Q\} \cup \Pi, \ell').$$

Proof. We use the map τ from (3). By the construction of the map, if some value only appears in one claim event in $\tau(\alpha)$, it will only appear in one claim event in α as well. This proves the theorem for *session-unique*.

As for data agreement, if events for some cast exist in $\tau(\alpha)$ with the correct value, they will also exist in α. This proves the theorem for *data-agree*. □

By definition of *session* and *wsession*, the preceding two theorems give conditions under which these two properties are preserved. This is expressed in the following

Corollary 39. *Let P, Q be protocols such that P establishes c, d is a parameter in all roles of Q and $P \cdot Q$ is defined. Let Π be a set of protocols, $P, Q \notin \Pi$. Let ℓ be the label of a (weak) session claim event in P or Q, and let ℓ' be the corresponding label in $P \cdot Q$. Then*

$$\mathrm{sat}(\{P\langle c\rangle, Q\langle c!/d\rangle\} \cup \Pi, \ell) \Rightarrow \mathrm{sat}(\{P \cdot Q\} \cup \Pi, \ell').$$

Proof. We use Theorem 37 and Theorem 38 to establish that the requisite *secret*, *session-unique* and *data-agree* claims hold, and the result follows. □

4 Mobile WiMAX

In this section we apply our framework to a handful of protocols from the security sublayer of the IEEE 802.16-2005 amendment [27] of the IEEE 802.16-2004 standard [26], commonly and in the following referred to as (mobile)

WiMAX. The aim is not a complete verification of WiMAX as this would constitute a research topic of its own. Instead, we use WiMAX to illustrate our framework on a real-world protocol suite and as a measure for our progress towards the goal of a comprehensive theory of protocol verification. We stress that even in the limited setting we consider, our methods are strong enough to draw useful conclusions about protocol design and security flaws.

4.1 Introduction

The IEEE 802.16-2004 standard specifies the air interface of fixed broadband wireless access systems supporting multimedia services in local and metropolitan area networks. The 802.16-2005 amendment addresses mobility of subscriber stations and features new security protocols.

A brief overview of the communication between a mobile station and base station follows. The communication starts at the mobile station's network entry with the *Ranging* protocol. Its purpose is to set up physical communication parameters and assign a basic connection identifier to the requesting mobile station. This protocol is periodically executed later to re-communicate the physical communication parameters. Next, a *Registration* protocol is carried out in order to allow the mobile station into the network. During this protocol, the base station and mobile station's security capabilities are negotiated. The base station and mobile station can agree on unilateral or mutual authentication, or no authentication at all, and on a variety of key management protocols. The key management protocols are periodically repeated to update the traffic encryption keys. The entire authentication chain is repeated on a less frequent basis. Once the traffic encryption keys are established, user data protocols start. To avoid service interruptions, traffic encryption keys have overlapping lifetimes.

The authentication and key management protocols are specified in the security sublayer of WiMAX. The security sublayer is meant to provide subscribers with privacy and authentication and operators with strong protection from theft of service [27, Chapter 7]. It employs an authenticated client/server key management protocol in which the base station controls distribution of keying material to the mobile station.

The security sublayer consists of two component protocols, an encapsulation protocol for securing packet data across the network and a key management protocol providing the secure distribution of keying data from the base station to the mobile station. In the following sections we will focus on the key management protocol. Through this key management protocol, the base station and mobile station are to synchronize keying data and the base station

is meant to use the protocol to enforce conditional access to network services.

The overall security goals mentioned in the specification are "no theft of service" for the operator of the base station and "confidentiality" for the user of the mobile station. Confidentiality in WiMAX is defined to be "privacy" and "authenticity" [27, Chapter 7, footnote 6]. The specification gives vague ideas for the security properties that the subprotocols have, by calling them for instance authentication protocols or key update protocols. But for a thorough security analysis, precise security claims need to be made for each subprotocol and the relation between the security properties achieved by the subprotocols and the overall security goals needs to be understood. We are addressing these issues in the following section.

4.2 Key management in the Security Sublayer

The privacy key management (PKM) component of the security sublayer consists of authentication and key establishment protocols. Depending on the negotiated security capabilities, the IEEE-802.16-2004 PKM protocols or the new PKM version 2 protocols will be executed. In the following security analysis, we consider the sequence of the three PKM version 2 protocols *PKMv2 RSA*, *PKMv2 SA-TEK*, and *PKMv2 Key*. *PKMv2 RSA* authenticates the base station (*bs*) and mobile station (*ms*) and establishes a shared secret which is used by *PKMv2 SA-TEK* and *PKMv2 Key* to secure the exchange of traffic encryption keys (TEKs). WiMAX does not explicitly state what the security claims of these three protocols are. As indicated in the Introduction, it is stated that the sequential composition of the three protocols achieves strong authentication and privacy for the mobile station, and strongly protects the base station from theft of service. Furthermore, it is implicitly stated that the established keys are shared secrets and *PKMv2 RSA* is called a mutual authentication protocol.

We are making these properties more precise by imposing the following requirements on the composition of the three protocols. In order to provide "strong protection against theft of service" [27, Chapter 7] for the base station, the client station has to be strongly authenticated at the end of the protocol composition, i.e. the role of the base station has to satisfy the *i-synch* claim and all key material must be secret. Note that if the *i-synch* claim is true at the end of the protocol composition then we are guaranteed that every message read by *bs* up to that point has been sent by *ms* and exactly matches the message sent by *ms*. Thus no theft of service can have occurred up to that point. In order to provide privacy and authenticity for the subscriber station, we demand that the base station is strongly authenticated at the end of the protocol composition, and that all session keys and symmetric encryption keys

are secret and unique.

We argue now that the following security properties for the three protocols and their sequential composition need to be fulfilled and prove in the next section that these properties indeed imply our set security goals. Since *PKMv2 RSA* needs to authenticate *ms* and *bs* and establish a shared secret which is to be used as a key later on, it has to satisfy *i-synch* for both roles and *session* for the shared secret. Since *PKMv2 SA-TEK* and *PKMv2 Key* need to establish further keys while keeping up the authentication property between *ms* and *bs*, they both need to satisfy *synch* for both roles, *session* for the shared secret, and at least *wsession* for the traffic encryption keys[5]. This implies in particular that the shared secret and traffic encryption keys have to be secret in all three protocols. We summarize these requirements in Table 2. It corresponds to the summary of verified security properties in Section 4.3.4.

Protocol	Security Properties
PKMv2 RSA	*i-synch*, *session*(*pPAK*)
PKMv2 SA-TEK	*i-synch*, *session*(*pPAK*), *wsession*(*TEK*)
PKMv2 Key	*synch*, *session*(*pPAK*), *wsession*(*TEK*)

Table 2
Required security properties for the PKMv2 subprotocols in WiMAX.

Before we start the description of the protocols, some technical remarks on our WiMAX model are in order. We will restrict ourselves to non-handover scenarios with unicast communication, leaving out multicast and broadcast communication, mesh communication, and group security. We further restrict ourselves to one Security Association as opposed to a list of several security associations offered by the base station. This is simply for convenience as it implies that there is only one pair of TEK keys instead of several pairs identified by SAID's and managed by parallel sessions of the *PKMv2 Key* protocol.

In our model, we have simplified messages by omitting irrelevant terms and headers. In particular, the fact that all messages in WiMAX are formatted using a type/length/value (TLV) scheme, we show only implicitly by giving appropriate names to identifiers. Entries in a TLV list are called attributes. WiMAX specification states that both roles silently discard all messages that do not contain all required attributes and skip over unknown attributes. Note that the use of TLVs together with signatures and hashed message authentication codes in WiMAX also implies that we may disregard type flaw attacks. Finally, we model hash functions as encryptions with a special public key, known by all parties, whose inverse key is not known by anyone. We thus

[5] Since *wsession* and *data-agree* imply *session*, the use of the *TEK*'s in user data protocols by both roles will automatically imply the *session* claim at that point.

write $\{\!|\, m \,|\!\}_{h(salt)}$ for the message m to which a message authentication code has been attached. Private key signatures will be indicated by $\{\!|\, m \,|\!\}_{sk(ms)}$ and $\{\!|\, m \,|\!\}_{sk(bs)}$.

4.2.1 PKMv2 RSA

The *PKMv2 RSA* protocol (Figure 4) is the initial mutual authentication protocol. It is repeated periodically to update the *pPAK*. Its purpose is to establish a shared secret *pPAK* (called pre-PAK in WiMAX) between *ms* and *bs*. The shared secret is used to derive the authentication key from which the keys for hashed message authentication codes and symmetric encryptions are derived.

According to specification, the *Request* message consists of *MS-Rnd*, *MS-Crt*, *SAID* and a signature over a SHA-1 hash of these fields using *ms*'s secret key, whose corresponding public key *bs* learns from *MS-Crt*. We model this by sending *MS-Crt* outside of $\{\!|\, \ldots \,|\!\}_{sk(ms)}$. The same applies to *BS-Crt* in the *Reply* and *Reject* messages.

4.2.2 PKMv2 SA-TEK

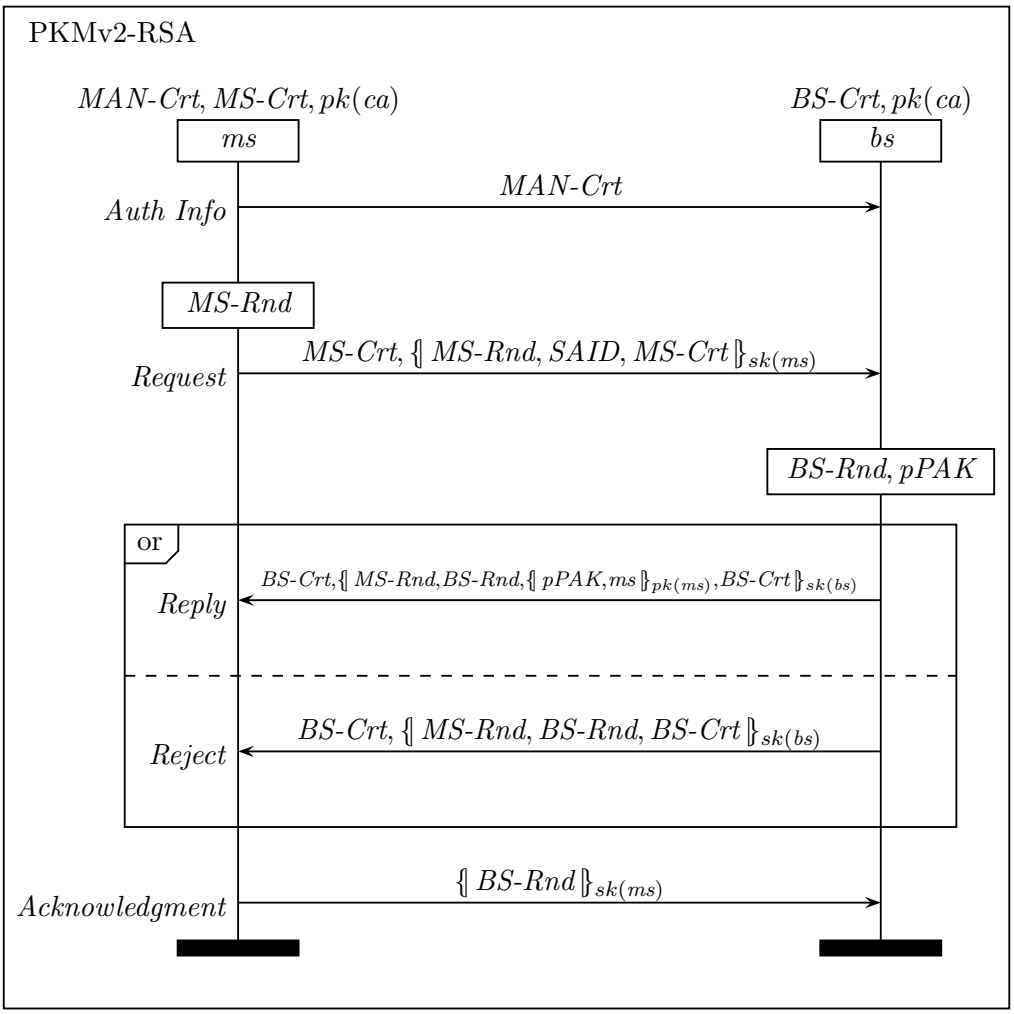
The *PKMv2 SA-TEK* protocol (Figure 5) is a three-way handshake protocol which follows the *PKMv2 RSA* protocol. Its purpose is to update the traffic encryption keys if they already exist. The protocol's messages are authenticated by hashing them with keys derived from *pPAK* established by the *PKMv2 RSA* protocol. Different keys are used for uplink and downlink traffic. The base station sends a challenge to the mobile station, which the mobile station repeats in its request for updated key material and thus proves liveness and knowledge of the shared secret (by using the derived *HMAC* key) to the base station. The base station answers then with updated key material, repeating the mobile station's nonce.

4.2.3 PKMv2 Key

The *PKMv2 Key* protocol (Figure 6) allows the mobile station to obtain the most recent *TEK* key from the base station.

4.3 Applying the Framework

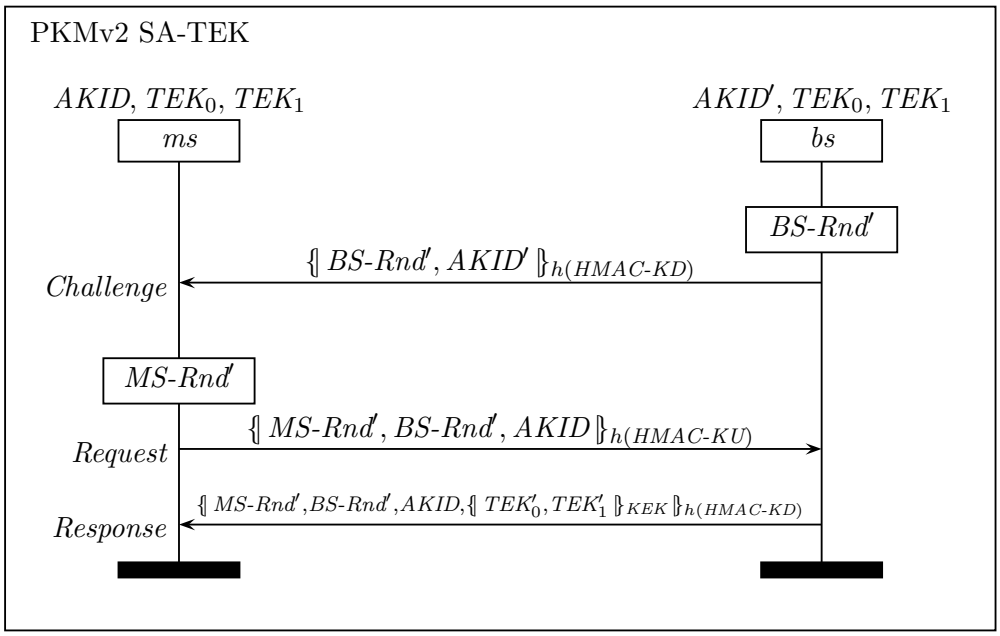
We begin by analyzing the three protocols described in the previous section in isolation and then we apply our theorems to their sequential composition. To facilitate later exposition, we will, during the course of the analysis, simplify



MAN-Crt	manufacturer's certificate
MS-Crt	mobile station's certificate $\{\!\mid ms, pk(ms) \mid\!\}_{sk(man)}$
BS-Crt	base station's certificate $\{\!\mid bs, pk(bs) \mid\!\}_{sk(ca)}$
SAID	security association Id, equal to *BCID*
MS-Rnd	mobile station's nonce
BS-Rnd	base station's nonce
pPAK	preliminary primary authentication key,
	actually another nonce created by the base station

Fig. 4. The *PKMv2 RSA* protocol.

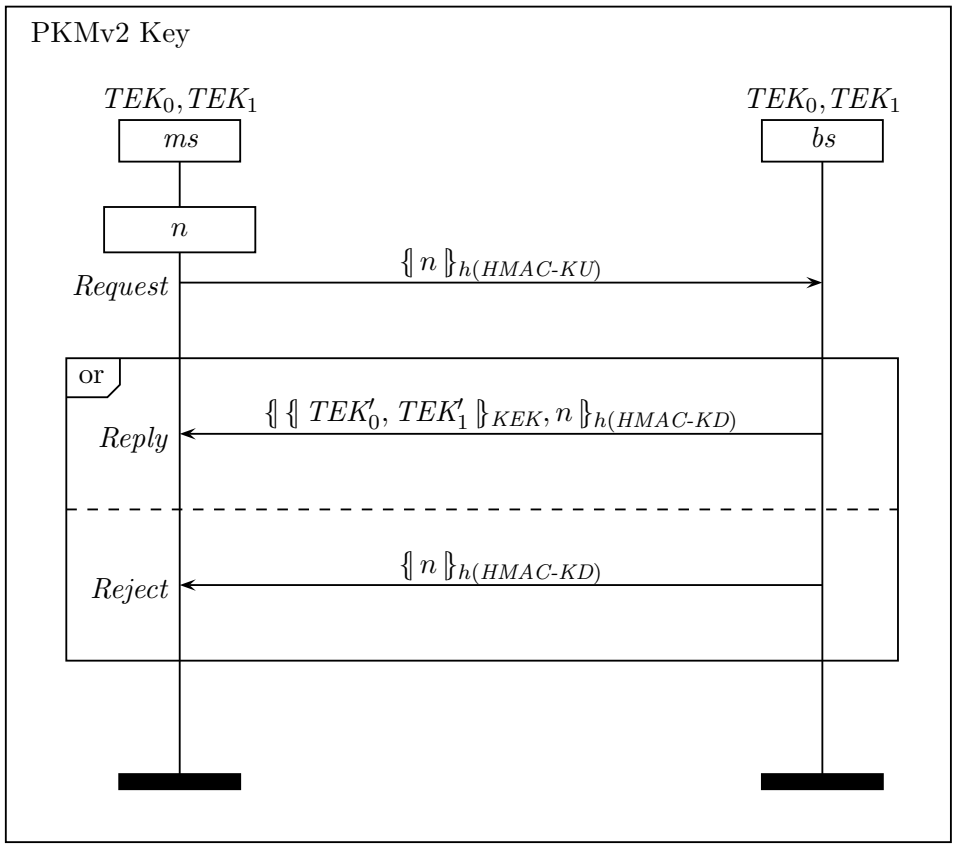
the protocols presented above. Since the aim of this work is not a careful and formal analysis of these short subprotocols, we will reason on an informal level for clarity, backed up by the automated verification tool Scyther. After that,



PAK	Primary authentication key derived from *pPAK* as follows: $PAK = keyedhash(pPAK, ms \mid bs)$.
AK	Authentication key derived from *PAK* as follows: $AK = keyedhash(PAK, ms \mid bs \mid PAK)$.
KEK	key encryption key derived from *AK* as follows: $HMAC\text{-}KU \mid HMAC\text{-}KD \mid KEK = keyedhash(AK, ms \mid bs)$.
HMAC-KD	HMAC key derived from *AK* for authenticating downlink communication
HMAC-KU	HMAC key derived from *AK* for authenticating uplink communication
MS-Rnd′	mobile station's nonce
BS-Rnd′	base station's nonce
AKSN	*AK* sequence number, essentially a 2 bit counter
AKID	*AK* Id: $AKID = keyedhash(AK, AKSN \mid ms \mid bs)$
AKID′	*AK* Id of new AK if re-authenticating

Fig. 5. The *PKMv2 SA-TEK* protocol.

we will analyse the composition in detail.



TEK_0	Older traffic encryption key
TEK_1	Newer traffic encryption key
TEK'_0, TEK'_1	Updated traffic encryption keys, replacing TEK_0, TEK_1, respectively [6]
KEK	Key Encryption Key (see above)

Fig. 6. The *PKMv2 Key* protocol.

4.3.1 Analysis of PKMv2 RSA

We analyze the *PKMv2 RSA* protocol without the *Auth Info* message which according to specification is only being sent right after *Ranging* and never again. We first consider *i-synch* for *ms* and *bs* and then the *session* property for *pPAK*.

4.3.1.1 *i-synch*. Since we have a choice in the third message between *Reply* and *Reject*, we will first analyze the branch with *Reply*, then the branch with *Reject*.

Reply. Note that the structure of *Request*, *Reply*, *Acknowledgment* is similar

to the standard X.509 protocol, except for the last message where the identity of *bs* is missing. As a consequence, it suffers from a man-in-the-middle attack causing *bs* to not synchronize. The intruder executes the man-in-the-middle attack by taking advantage of an *ms* trying to connect to the intruder and redirecting those messages to a *bs*. In [43] the authors describe a similar, but slightly more complicated attack, in which the intruder uses two runs of *ms* to impersonate *ms* to *bs*. Both attacks can be found using Scyther.

The attacker can not, however, impersonate *bs* to *ms*, and the *ms* role in fact still synchronizes and the *pPAK* remains secret. It is also interesting to note that *Request*, *Reply*, *Acknowledgment* followed by *PKMv2 SA-TEK* has agreement (an authentication property slightly weaker than synchronisation, see [16]) for both *ms* and *bs*. However, we still consider the lack of *bs*'s identity in *Acknowledgment* a design flaw, since it is supposed to be a "mutual authentication" protocol according to specification and hence, in its current form, breaks a modular design principle: small changes in other protocols (for instance *PKMv2 SA-TEK*) could break the security of the entire composition.

For future reference, we write P to denote the protocol consisting of *Request*, *Reply*, and *Acknowledgment*' where *Acknowledgment*' is the message $\{\!|\, \textit{BS-Rnd}, bs \,|\!\}_{sk(ms)}$ from *ms* to *bs*. Furthermore, we will denote the *pPAK* in P simply by c. P has injective synchronization for both roles.

Reject. As in the branch analyzed above, the structure *Request*, *Reject*, *Acknowledgment* resembles the X.509 standard. Here however, both *Reject* and *Acknowledgment* are missing the recipient's ID. Therefore neither *bs* nor *ms* synchronize (the weaker agreement notion is not satisfied either). A *pPAK* is not being sent, thus the secrecy claim is void. The fact that *ms* does not synchronize can be abused for a denial of service attack.

We amend the flaws pointed out above by considering P instead of *PKMv2-RSA*. Note the absence of a *Reject* message from P. A *Reject* message sent from *bs* to *ms* terminates the protocol. In order for *ms* to communicate with *bs* it has to start over with the *Ranging* protocol. For this reason, we will simplify the analysis of the composition in Section 4.3.4, without affecting any of the security properties we are interested in, if we consider the protocol without the *Reject* message; instead of a send event corresponding to the *Reject* message, the run of the *bs* role ends.

The specification of protocol P is given below. For brevity, we omit the typing of identifiers and use the shorthand for the message contents as displayed in Figure 4. The descriptive labels of the claim events are inserted for further reference.

$$
\begin{aligned}
P(ms) = {} & create_{P1}(ms) \cdot send_{P2}(ms, bs, Request) \cdot read_{P3}(bs, ms, Reply) \cdot \\
& send_{P4}(ms, bs, \{\!|\, BS\text{-}Rnd, bs \,|\!\}_{sk(ms)}) \cdot claim_{i\text{-}synch(P,ms)}(ms, i\text{-}synch) \cdot \\
& claim_{session(P,ms,c)}(ms, session, c) \cdot end_{P5}(ms) \\
P(bs) = {} & create_{P6}(bs) \cdot read_{P2}(ms, bs, Request) \cdot send_{P3}(bs, ms, Reply) \cdot \\
& read_{P4}(ms, bs, \{\!|\, BS\text{-}Rnd, bs \,|\!\}_{sk(ms)}) \cdot claim_{i\text{-}synch(P,bs)}(bs, i\text{-}synch) \cdot \\
& claim_{session(P,bs,c)}(bs, session, c) \cdot end_{P7}(bs)
\end{aligned}
$$

In the remainder, we will use the abbreviation $i\text{-}synch(P)$ to stand for the combination of $i\text{-}synch(P, ms)$ and $i\text{-}synch(P, bs)$. We will use similar abbreviations for the other claims and protocols.

Using Scyther, we prove synchronisation. Given the fact that P satisfies the *loop-property* from [16], we establish

$$\text{sat}(\{P\}, \{i\text{-}synch(P)\}).$$

4.3.1.2 *session.* We use Scyther to verify that *pPAK* is secret. The *data-agree* property is satisfied because of injective synchronization, and the fact that *pPAK* is part of a message causally preceding the synchronization claims. Finally, *session-unique* is also satisfied because of injective synchronization and the fact that *pPAK* is a constant in one of the roles appearing only in one send event, accompanied, within a signature, by the recipient's nonce.

The established result is

$$\text{sat}(\{P\}, \{session(P, c)\}).$$

4.3.2 Analysis of PKMv2 SA-TEK

We call Q the protocol obtained from *PKMv2 SA-TEK* by introducing the parameter d, which will obtain its value from the constant c produced by protocol P. The parameter d will hold the shared secret established in P from which the *HMAC* and *KEK* keys are derived. Furthermore, the collection of *TEK* keys will be denoted by e in Q. Thus, Q is up to renaming of constants equivalent to *PKMv2 SA-TEK*.

We insert session claim events for d, weak session claim events for e, and injective synchronization at the end of both roles, with labels $session(Q, ms, d)$, $session(Q, bs, d)$, $wsession(Q, ms, e)$, $wsession(Q, bs, e)$, $i\text{-}synch(Q, ms)$, and

i-synch(Q, bs). This yields the following description of protocol Q (assuming $h1$, $h2$, and $h3$ are distinct hash functions).

$$
\begin{aligned}
Q(ms) = \; & create_{Q1}(ms) \cdot read_{Q2}(bs, ms, \{\!|\, \mathit{BS\text{-}Rnd}', \mathit{AKID}' \,|\!\}_{h1(d)}) \cdot \\
& send_{Q3}(ms, bs, \{\!|\, \mathit{MS\text{-}Rnd}', \mathit{BS\text{-}Rnd}', \mathit{AKID} \,|\!\}_{h2(d)}) \cdot \\
& read_{Q4}(bs, ms, \{\!|\, \mathit{MS\text{-}Rnd}', \mathit{BS\text{-}Rnd}', \mathit{AKID}, \{\!|\, e \,|\!\}_{h3(d)} \,|\!\}_{h1(d)}) \cdot \\
& claim_{\mathit{i\text{-}synch}(Q,ms)}(ms, \mathit{i\text{-}synch}) \cdot claim_{\mathit{session}(Q,ms,d)}(ms, \mathit{session}, d) \cdot \\
& claim_{\mathit{wsession}(Q,ms,e)}(ms, \mathit{wsession}, e) \cdot end_{Q5}(ms) \\
Q(bs) \; = \; & create_{Q6}(bs) \cdot send_{Q2}(bs, ms, \{\!|\, \mathit{BS\text{-}Rnd}', \mathit{AKID}' \,|\!\}_{h1(d)}) \cdot \\
& read_{Q3}(ms, bs, \{\!|\, \mathit{MS\text{-}Rnd}', \mathit{BS\text{-}Rnd}', \mathit{AKID} \,|\!\}_{h2(d)}) \cdot \\
& send_{Q4}(bs, ms, \{\!|\, \mathit{MS\text{-}Rnd}', \mathit{BS\text{-}Rnd}', \mathit{AKID}, \{\!|\, e \,|\!\}_{h3(d)} \,|\!\}_{h1(d)}) \cdot \\
& claim_{\mathit{i\text{-}synch}(Q,bs)}(bs, \mathit{i\text{-}synch}) \cdot claim_{\mathit{session}(Q,bs,d)}(bs, \mathit{session}, d) \cdot \\
& claim_{\mathit{wsession}(Q,bs,e)}(bs, \mathit{wsession}, e) \cdot end_{Q7}(bs)
\end{aligned}
$$

Again, we verify injective synchronization and secrecy for d and e using Scyther. Note that the verification has to be done for the trace restriction $Q\langle \mathit{session}/d \rangle$. Session trace restrictions can be simulated in Scyther using technical tricks, but are expected to be supported natively in the future.

The *session-unique* property follows from arguments analogous to the ones shown for c in P and the session trace restriction for both d and e. Data agreement for d follows from injective synchronization and the appearance of d in a message causally preceding the *i-synch* claim for both roles. For e we do not get data agreement for the *bs* role, since e is sent in the last message for which *bs* has no guarantee that *ms* received it.

We have shown

$$\mathrm{sat}(\{Q\langle \mathit{session}/d \rangle\}, \{\mathit{session}(Q, d), \mathit{wsession}(Q, e), \mathit{i\text{-}synch}(Q)\})$$

4.3.3 Analysis of PKMv2 Key

Similarly to the previous two protocols, we will let R denote the protocol obtained from *PKMv2 Key* by denoting the *HMAC* keys by d, the old *TEK* keys by e and the new *TEK* keys by e'. We let R only consist of the *Request* and *Reply* messages, since the alternative has exactly the same security properties.

We have *i-synch* for the *ms* role, shown by applying Scyther and using the

loop property, but only *synch* for *bs*.

The *session* property for d and *wsession* property for e' can be shown in exactly the same manner as for the Q protocol.

$$
\begin{aligned}
R(ms) = {} & create_{R1}(ms) \cdot send_{R2}(ms, bs, \{\!|\, n \,|\!\}_{h2(d)}) \cdot \\
& read_{R3}(bs, ms, \{\!|\, \{\!|\, e' \,|\!\}_{h3(d)}, n \,|\!\}_{h1(d)}) \cdot \\
& claim_{i\text{-}synch(R,ms)}(ms, i\text{-}synch) \cdot claim_{session(R,ms,d)}(ms, session, d) \cdot \\
& claim_{wsession(R,ms,e')}(ms, wsession, e') \cdot end_{R5}(ms) \\
R(bs) = {} & create_{R6}(bs) \cdot read_{R2}(ms, bs, \{\!|\, n \,|\!\}_{h2(d)}) \cdot \\
& send_{R3}(bs, ms, \{\!|\, \{\!|\, e' \,|\!\}_{h3(d)}, n \,|\!\}_{h1(d)}) \cdot \\
& claim_{synch(R,bs)}(bs, synch) \cdot claim_{session(R,bs,d)}(bs, session, d) \cdot \\
& claim_{wsession(R,bs,e')}(bs, wsession, e') \cdot end_{R7}(bs)
\end{aligned}
$$

Abbreviating the claim labels, we obtain

$$
\begin{aligned}
\mathrm{sat}(\{R\langle session/d\rangle\}, \{&i\text{-}synch(R, ms), synch(R, bs), \\
&session(R, d), wsession(R, e')\}).
\end{aligned}
$$

4.3.4 The composition

In the preceding three subsections we have established that

$$
\begin{aligned}
&\mathrm{sat}(\{P\}, \{i\text{-}synch(P), session(P, c)\}) && (4)\\
&\mathrm{sat}(\{Q\langle session/d\rangle\}, \{session(Q, d), wsession(Q, e), i\text{-}synch(Q)\}) && (5)\\
&\mathrm{sat}(\{R\langle session/d\rangle\}, \{i\text{-}synch(R, ms), synch(R, bs), \\
&\qquad\qquad session(R, d), wsession(R, e')\}) && (6)
\end{aligned}
$$

The methodology for the verification of these facts is standard. In what follows, we apply the collection of theorems in our framework to show how the established properties imply correctness of the composed protocol $P \cdot Q \cdot R$.

Note that P, Q, and R are mutually strongly independent since the messages, as stated in Section 4.2, are TLV encoded, the signatures or hashed message authentication codes are made over the entire message, and the message structures are different in the three protocols. More precisely, protocol P is independent from Q and R because all messages in P are signed by private keys and hence the signatures will not be accepted by either role in protocols

Q and R, their messages being authenticated using the shared secret keys. Protocols Q and R are strongly independent, since they don't have a message in common in which all required attributes are identical.

Using strong independence, we can now deduce that P followed by Q satisfies injective synchronization, session, and weak session as follows.

By Theorem 32 and equations (4) and (5), we can preserve the *i-synch* property for P, Q. By Corollary 33, c and d keep the *session* property, and e the *wsession* property.

Therefore, we obtain

$$\begin{aligned}\mathrm{sat}(\{P\langle c\rangle, Q\langle c/d\rangle\}, \{&\textit{i-synch}(P), \textit{i-synch}(Q),\\ &\textit{session}(P,c), \textit{session}(Q,d), \textit{wsession}(Q,e)\}),\end{aligned}$$

and using Theorems 36 (to obtain *i-synch*), and Corollary 39 (*wsession* and *session*) we get

$$\mathrm{sat}(\{P \cdot Q\}, \{\textit{i-synch}(PQ), \textit{session}(PQ,c), \textit{wsession}(PQ,e)\}). \tag{7}$$

Here we assume that in the composed protocol $P \cdot Q$ roles *bs* and *ms* are extended with appropriate claim events. We use the three labels $\textit{i-synch}(PQ)$, $\textit{session}(PQ,c)$, and $\textit{wsession}(PQ,e)$ to refer to these claims.

Next, by Theorem 32 we establish from (6) and (7) injective synchronization for both roles and by Corollary 33 *session* for c and *wsession* for e:

$$\begin{aligned}\mathrm{sat}(\{P \cdot Q\langle c\rangle, R\langle c/d\rangle\}, \{&\textit{i-synch}(PQ),\\ &\textit{session}(PQ,c), \textit{wsession}(PQ,e),\\ &\textit{i-synch}(R, \textit{ms}), \textit{synch}(R, \textit{bs}),\\ &\textit{session}(R,d), \textit{wsession}(R,e')\})\end{aligned}$$

Using Theorem 36 and Corollary 39 one more time, we can show that the entire composition satisfies injective synchronization and (weak) session secrecy for the shared secret c and the traffic encryption key e:

$$\begin{aligned}\mathrm{sat}(P \cdot Q \cdot R, \{&\textit{i-synch}(PQR), \textit{session}(PQR,c),\\ &\textit{wsession}(PQR,e), \textit{wsession}(PQR,e')\})\end{aligned}$$

These are exactly the overall security properties we have formulated in Section 4.2. As we have shown in that section, these security properties are a precise interpretation of the security goals stated in the WiMAX specification.

We have verified that the sequential composition of the key management protocols *PKMv2 RSA*, *PKMv2 SA-TEK*, and *PKMv2 Key* satisfies strong authentication and session secrecy for keys derived from the shared secret and for traffic encryption keys. However, in order to achieve this verification, we had to first formulate precise security properties for the subprotocols and their composition, based on our interpretation of the rather vague security goals specified in WiMAX.

While we have shown that our strong authentication property *i-synch* holds for the protocol composition as stated, it has to be noted that a composition consisting of repeated iterations of the *PKMv2 Key* protocol would fail to satisfy this property, due to replay attacks. In practice, such attacks would only be a nuisance, not a security threat, since *secret* would still hold true for all keys, and such an active intruder would not be able to learn anything more than a passive, listening, one. Thus, in future work, the WiMAX protocols will be analyzed with an appropriately weakened notion of authentication.

Although several studies on WiMAX security have appeared, none of these offer a precise and compositional verification of the WiMAX key management protocols. Closest to our work is a preliminary study in [30], which sketches the steps towards a compositional verification. An analysis of some protocols from the 2001 version of the WiMAX standard is conducted in [28]. Their main observation is that the old protocols achieve unilateral authentication, while mutual authentication of *bs* and *ms* is required. The proposed fixes clearly found their way into the current standard. However, the protocols are studied in isolation and the authors did not apply formal reasoning to prove the fixed protocols correct. Xu et al. [43,44] analyzed several isolated protocols from the current WiMAX standard. Through informal reasoning, they discovered the attack mentioned in Section 4.3.1 and proposed a fix. The modified protocol is proved correct by using the BAN-logic [9], which is known to be incomplete with respect to insider attacks. Finally, we mention an analysis of the current WiMAX standard using the TLA+ logic in [44]. The authors study the composition of the three mentioned key management protocols as one single protocol. Exploiting symmetry reduction techniques, they manage to apply the TLC model checker to verify the composed protocol. However, their focus is not on the key management properties that we investigate (such as authentication and secrecy), but on detecting a class of denial-of-service attacks. In order to validate liveness, they focus on the state machines underlying the protocols.

5 Related Work

In this section we address work related to the composition of security protocols. We discuss strand spaces, as well as the more recent Protocol Composition Logic, in some detail below. Afterwards we address related theoretical results, and discuss some attempts at the verification of composed protocols.

5.1 Strand Spaces

Within a modified version of the Strand Spaces framework [41], called the Mixed Strand Spaces model [42], some results about compositionality have been proven. In [22], a disjoint encryption theorem is proven. This theorem states that if two protocols have sufficiently different encrypted messages (at the trace/run level), composing them in parallel will not introduce new attacks. In terms of methodology, this work is closely related to ours: given *any* two correct protocols, what abstract properties should they satisfy, in order to ensure that their composition is correct?

The main differences between their approach and ours, are that (1) they only consider parallel composition, and (2) verification of the disjoint encryption property has to be done at the level of traces. With respect to the first item, we note that the approach does not allow for the decomposition of large sequential protocols into smaller ones, as can be done with the chaining theorems presented here. Similarly, there are no concepts such as session-secrecy. Consequently, the disjoint encryption approach cannot be used for the compositional verification of strongly dependent subprotocols such as those present in WiMAX, for instance.

The second item represents a more significant drawback of the approach. The verification of disjoint encryption has to be performed at the trace level of the composed protocols. For some protocols, this can be easily, but nevertheless manually, deduced from the protocol specification, but in many other cases (e.g. where session keys are used in protocols) the only way to verify that the disjoint encryption property holds is by inspecting the traces of the composed protocol. Therefore, in such cases there is no expected improvement on the more traditional approach of, for example, model checking all traces of the composed protocol.

As a more subtle drawback, the proof given for the disjoint encryption theorem assumes that the security properties do not include ordering constraints. Thus, it is not immediately possible to apply the theorem for the verification of strong authentication properties, such as e.g. the synchronization property.

One advantage of the approach is that their results also hold for protocols that include tickets, something that we have explicitly excluded here.

5.2 Protocol Composition Logic

One of the most significant theoretical results in the work on Protocol Composition Logic (PCL) [17] is a strategy for dealing with protocol composition. The basic idea is to prove the protocols correct in isolation by constructing a correctness proof in the logic. Certain so-called invariants are then identified in the correctness proofs, such that protocol correctness follows from these invariants only. If these invariants are not violated by the other protocols, it is an easy consequence that correctness is retained under composition, since correctness follows from the invariants alone.

In contrast to this very general strategy, our composition theorems identify specific classes of protocols that can be composed in certain ways. The advantage of our approach is that it is highly amenable to automatic verification (as demonstrated in Section 4), especially when combined with the trivially verifiable strong independence property. In contrast, the full generality of proof derivation in PCL seems difficult to automate.

It is easy to see that our notion of strong independence is a stronger requirement than the invariants used for composition in PCL. Nevertheless, strong independence is trivial to verify, hence highly suited for an automatic verification strategy. It is possible that composition theorems similar to ours, based on strong independence, could be recovered in the PCL framework.

The PCL invariant approach can deal with cases that our notion of independence cannot. The reason for this is that independence only considers ciphertext terms and their origins, while PCL invariants cover more general statements. Conversely, since independence is verified for traces while PCL invariants are verified over so-called basic sequences, it is not immediately obvious that the PCL approach can deal with every case our independence notion can deal with. Investigating this relationship is an interesting topic for future research.

We believe that many ideas and techniques used in PCL can be reused in our framework. The techniques used to identify and verify invariants could possibly be used to prove independence. Due to the highly manual nature of the PCL compositionality strategy, we expect any such work to be complementary to the theory developed in this paper, to be used only when automatic techniques fail.

An alternative approach is taken in the PDa tool [3]. In this tool, an ax-

iomatic theory is set up to reason about protocol refinement and composition. The tool uses ideas from PCL to reason about invariants. The tool can provide automatic discharging of simple proofs. However, the user has to provide sufficiently strengthened invariants to allow for compositional proofs. The axiomatic theory does not have a notion of run (or process or thread), similar to e.g. BAN logic, and as a result only very weak notions of authentication can be considered.

5.3 Related theoretical results

The complex problem of compositionality has been approached from a variety of angles. Many of these approaches are restricted to weak forms of authentication, such as [7,8,33]. When only such weaker forms are considered, compositionality results can be achieved on the basis of simpler challenge-response mechanisms within a protocol, similar to the authentication tests from [23]. The existence of these mechanisms in a protocol does not ensure synchronization, or even agreement.

Other approaches have considered secrecy, e.g. [29]. Here a notion of secrecy is defined within the context of stream-processing functions. Using the notion of an m-secrecy protecting process, a result is given that states that two such processes can be safely composed. Furthermore, it is shown that such a process remains secrecy-protecting under refinement. Similar to the Strand Spaces approach, it is unclear how one can establish that a process (or protocol) satisfies the required conditions for the stated theorems.

In the area of information flow analysis, which is related to the secrecy-only approach, there are a number of results and supporting tools, e.g. [20, 21, 25, 29, 34]. However, because of fundamental differences in the underlying models, these results cannot be used for the compositional analysis of security protocols such as WiMAX.

In [10] the observation is made that the correctness of security protocols depends on the assumptions on the environment. In the wrong environment, or in the context of specific protocols, seemingly secure protocols are incorrect. The authors give no specific conditions or properties.

Over the past decades compositional verification has received quite some attention from the process algebra community and was applied successfully in the verification of complex concurrent systems (for an overview of these techniques, see e.g. [39]). However, these techniques do not seem to carry over easily to the process algebras developed especially for security protocols, such as the spi calculus [1]. An attempt has been made in [6]. Here, compositionality is interpreted as a congruence property of a bisimulation-like relation

over several process operators. Although the authors provide compositional rules for (restricted) parallel and action-prefix composition, rules for general sequential composition are absent, making it impossible to apply this work to e.g. the WiMAX protocol suite. Moreover, the rule for parallel composition poses a very strong restriction on the set of processes that may run in parallel with any given process.

Furthermore, the security properties treated are secrecy and weak forms of authentication. It is not obvious how general protocol-centric properties and especially injectivity can be expressed by means of the bisimulation relation provided. Finally, we note that the proposed methodology has severe limitations with respect to the verification of actual protocols. As an example, the authors prove correctness of a version of the Wide Mouthed Frog protocol, which is obviously insecure in the standard setting for security protocols. This problem is due to the fact that their theory only supports the verification of fixed scenarios.

5.4 Verification of composed security protocols

In the area of protocol verification, it seems that the first attempt at verification of parallel subprotocols was made in [35], where the interaction between subprotocols was investigated manually.

An attempt at composing security protocol proofs within a theorem-proving environment was made in [40]. In this work, the authors construct compositional proofs for a specific protocol, in order to work towards a general theory. The conclusion of the authors is that even for a single protocol, the approach requires much manual work, and that scaling problems might cause this approach to be infeasible.

More recently fully automated verification of composed protocols was performed in [13] employing the same basic framework and tool used here. However, this verification, too, has been limited to protocols that are composed in parallel.

6 Conclusion

We make two significant contributions in this paper. First, we create a framework for easy verification of a large and useful class of security protocols built from smaller subprotocols. Second, we initiate a study of WiMAX, by applying our framework to the composition of three protocols from its security

sublayer. This is done by first verifying that the three protocols are independent and then analyzing each subprotocol in isolation. The results of this analysis are used to deduce properties about the composition of the subprotocols, consisting of the subprotocols running in parallel with suitable transfer of information. Consequently, this allows us to derive properties of the composed protocol.

We do not claim that our framework can deal with every possible security protocol. One important restriction is the requirement in many theorems that subprotocols are independent. This makes it difficult to use our framework for analysis of protocols that do not naturally split into independent subprotocols. As we have argued, protocol tags are a reasonably cost-effective way to design protocols that are amenable to analysis in our framework. WiMAX is just one example of protocols in which such techniques are in use today. We believe this is a very reasonable approach to future protocol design.

A significant advantage of our framework is ease of use. If we consider the WiMAX analysis, the Scyther tool automatically proves secrecy and synchronization for the subprotocols. Since session-uniqueness and data agreement claims have not yet been implemented in Scyther, a small amount of reasoning is needed to prove that the protocols have these properties in isolation. Once the properties are established, however, using the theorems to deduce the security properties of the composed protocol is essentially trivial. As the WiMAX analysis to some degree shows, it should be possible to verify protocols without an intimate knowledge of the underlying semantics described in Section 2, since a tool like Scyther (once it is suitably extended) can deal with the proofs needed at this level.

An interesting feature of our framework is that the theorem statements are not strongly connected to the underlying semantics. They are therefore in a sense independent of the semantics. Indeed, we believe the framework could be transferred to any other semantics powerful enough to express at least the notion of independence and the security properties, and which has a similar execution model.

In general, our theorems are tight in the sense that if any precondition is relaxed, the theorem is no longer true. Of course, some theorems could be extended in natural ways, and other theorems have many specialized variations. For the current work, we believe such extensions would add little value. Instead, such results should be proved as needed, slowly increasing the knowledge about how composition works.

In this work, we have defined the protocol-centric class of security properties and proved many theorems for that class. Likewise, we can define other classes of properties, for instance properties that only consider the intruder's memory.

Studying such classes of properties and proving theorems about them is an interesting future topic.

Another useful contribution in this paper is our definition of protocol independence. Currently, we have only described one way to achieve independence, namely protocol tags. There are several other ways one could imagine achieving independence, for instance through some notion of separate key infrastructures. One can also imagine other notions of independence that allow general theorems to be proved. Such notions would create new protocol design strategies and allow more protocols to be analyzed. We intend to continue our work on this topic.

The requirement in our semantics that variables only contain nonce run terms prevents us from expressing protocols using tickets in the semantics. The requirement is only essential for Theorem 26. A more significant problem is the fact that security properties such as synchronization or agreement do not make sense in the context of tickets, since some roles are by definition insensitive to the content of the tickets. An important topic for future work will be to extend our framework with new security properties and new theorems for ticket-based security protocols.

In our framework we discuss how to compose protocols. While sequential composition is the natural notion of protocol composition, there are other possible composition operators that are natural to discuss, such as the choice operator allowing one out of two protocols to run. Extending our framework with such operators and theorems to support reasoning with them is an important future topic.

As we have already noted, the Scyther tool does not have support for every security property we have defined, nor for every trace restriction. In the near future, we intend to extend Scyther with support for these security properties and trace restrictions. A related task is the creation of a new tool to formally verify reasoning in our framework. Essentially, this tool will use Scyther as a back-end to analyze the subprotocols, then it will verify that every theorem application is valid. This will allow automated verification of large protocols. As the body of theorems in our framework increases, so will the power of the tool when the theorems are added.

We have analyzed the security requirements of WiMAX and shown that a somewhat restricted variant of the protocol satisfies these requirements, all by reasoning in our framework and analyzing small subprotocols in isolation. We believe our study, though not complete, is a useful first step towards a complete analysis of the security requirements of WiMAX, as well as towards a verification of the entire protocol suite. In the future, we intend to work out a complete analysis of the security sublayer of WiMAX.

Today, most new protocols are not verified (in any sense of the word) when they are released, for example, as standards. We believe this is because today, verification of any sizable protocol is the exclusive province of the few skilled specialists and researchers working in the area. An important goal of current research is to remedy this problem. As the analysis of the WiMAX protocols show, our work is a significant first step towards a framework for security protocol analysis (with tool support) that could be used by engineers to verify protocols during design, allowing a proper security analysis of the protocol before release.

Acknowledgment We thank Eric Kaasenbrood for his help in understanding and modeling WiMAX. We also thank the anonymous reviewers whose comments have helped to improve this paper.

References

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

[2] J. Alves-Foss. Provably insecure mutual authentication protocols: The two-party symmetric encryption case. In *Proc. National Information System Security Conference*, pages 306–314, October 1999.

[3] M. Anlauff, D. Pavlovic, R. Waldinger, and S. Westfold. Proving authentication properties in the Protocol Derivation Assistant. In Pierpaolo Degano, Ralph Küsters, and Luca Vigano, editors, *Proceedings of FCS-ARSPA 2006*. ACM, 2006.

[4] 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 *Proc. Computer Aided Verification'05 (CAV)*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

[5] M. Backes, A. Datta, A. Derek, J.C. Mitchell, and M. Turuani. Compositional analysis of contract-signing protocols. *Theor. Comput. Sci.*, 367(1-2):33–56, 2006.

[6] M. Boreale and D. Gorla. On compositional reasoning in the spi-calculus. In M. Nielsen and U. Engberg, editors, *Proc. of 5th Intern.Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'02)*, volume 2303 of *LNCS*, pages 67–81. Springer, 2002.

[7] M. Bugliesi, R. Focardi, and M. Maffei. Authenticity by tagging and typing. In *FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 1–12, New York, NY, USA, 2004. ACM Press.

[8] M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of authentication protocols. In D. A. Schmidt, editor, *Proc. of the 13th European Symposium on Programming (ESOP)*, volume 2986 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2004.

[9] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Practical Cryptography for Data Internetworks*. IEEE Computer Society Press, 1996. Reprinted from the Proceedings of the Royal Society, volume 426, number 1871, 1989.

[10] R. Canetti, C. Meadows, and P. Syverson. Environmental requirements for authentication protocols. In M. Okada, B.C. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, editors, *Software Security – Theories and Systems, Mext-NSF-JSPS International Symposium, ISSS 2002*, volume 2609 of *Lecture Notes in Computer Science*, pages 339–355, Tokyo, Japan, 2002. Springer.

[11] C.J.F. Cremers. Scyther: Automatic verification of security protocols. `http://www.win.tue.nl/~ccremers/scyther/`.

[12] C.J.F. Cremers. Compositionality of security protocols: a research agenda. In F. Gadducci and M. ter Beek, editors, *VODCA 2004*, volume 142(3) of *ENTCS*, pages 99–110, Bertinoro, Italy, 2006.

[13] C.J.F. Cremers. Feasibility of multi-protocol attacks. In *Proc. of the first international conference on availability, reliability and security (ARES)*, pages 287–294, Vienna, Austria, April 2006. IEEE Computer Society Press.

[14] C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. PhD thesis, Eindhoven University of Technology, 2006. ISBN 90-386-0804-7. - ISBN 978-90-386-0804-4.

[15] C.J.F. Cremers and S. Mauw. Operational semantics of security protocols. In S. Leue and T.J. Systä, editors, *Scenarios: Models, Algorithms and Tools (Dagstuhl 03371 post-seminar proceedings, September 7–12, 2003)*, volume 3466 of *Lecture Notes in Computer Science*, pages 66–89, 2005.

[16] C.J.F. Cremers, S. Mauw, and E.P. de Vink. Injective synchronisation: an extension of the authentication hierarchy. *Theoretical Computer Science*, 367(1-2):139–161, November 2006. Special issue on ARSPA'05, (P. Degano and L. Vigano, eds.).

[17] A. Datta, A. Derek, J. C. Mitchell, and A. Roy. Protocol composition logic (pcl). *Electron. Notes Theor. Comput. Sci.*, 172:311–358, 2007.

[18] A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. Secure protocol composition. In *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 11–23, New York, NY, USA, 2003. ACM Press.

[19] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1983.

[20] R. Focardi and R. Gorrieri. Automatic compositional verification of some security properties. *Lecture Notes in Computer Science*, 1055:166–186, 1996.

[21] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Trans. Software Eng*, 23(9):550–571, 1997.

[22] J.D. Guttman and F.J. Thayer. Protocol independence through disjoint encryption. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000. `citeseer.ist.psu.edu/guttman00protocol.html`.

[23] J.D. Guttman and F.J. Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2001.

[24] C. He, M. Sundararajan, A. Datta, A. Derek, and J.C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM Conference on Computer and Communications Security*, pages 2–15. ACM, 2005.

[25] N. Heintze and J.D. Tygar. A model for secure protocols and their compositions. *IEEE Trans. Softw. Eng.*, 22(1):16–30, 1996.

[26] IEEE. standard 802.16-2004, 2004.

[27] IEEE. standard 802.16e-2005, 2005.

[28] D. Johnston and J. Walker. Overview of IEEE 802.16 security. *IEEE Security & Privacy*, 2(3):40–48, 2004.

[29] J. Jürjens. Composability of secrecy. In V. Gorodetski, V. Skormin, and L. Popyack, editors, *International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS 2001)*, volume 2052 of *Lecture Notes in Computer Science*, pages 28–38, St. Petersburg, May 2001. Springer.

[30] E. Kaasenbrood. WiMAX security - A formal and informal analysis. Master's project, Eindhoven University of Technology, Department of Mathematics and Computer Science, 2006.

[31] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In B. Christianson, B. Crispo, T.M.A. Lomas, and M. Roe, editors, *Proceedings of the 5th International Workshop on Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 91–104, London, UK, 1998. Springer.

[32] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[33] M. Maffei. Tags for multi-protocol authentication. In *Proc. of the 2nd International Workshop on Security Issues in Coordination Models, Languages, and Systems*, volume 128(5) of *Electronic Notes in Theoretical Computer Science*, pages 55–63. Elsevier ScienceDirect, August 2005.

[34] H. Mantel. On the composition of secure systems. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 88–101, Washington, DC, USA, 2002. IEEE Computer Society.

[35] C. Meadows. Analysis of the Internet Key Exchange Protocol using the NRL protocol analyzer. In *Proc. 20th IEEE Symposium on Security & Privacy*, pages 216–231. IEEE Computer Society, 1999.

[36] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proc. of DISCEX 2000*, pages 237–250. IEEE Computer Society Press, 2000.

[37] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(2):120–126, February 1978.

[38] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[39] W.P. de Roever, U. Hanneman, J. Hooiman, Y. Lakhneche, M. Poel, J. Zwiers, and F. de Boer. *Concurrency Verification. Introduction to Compositional and Noncompositional Methods*, volume 54 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 2001.

[40] O. Sheyner and J. Wing. Composing proofs of security protocols using Isabelle/IOA, August 2000. In proc. of the Theorem Proving for Higher Order Logics (TPHOLs) workshop, short paper.

[41] F.J. Thayer, J.C. Herzog, and J.D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 1998 IEEE Symposium on Security and Privacy*, pages 66–77, Oakland, California, 1998.

[42] F.J. Thayer, J.C. Herzog, and J.D. Guttman. Mixed strand spaces. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, pages 72–82. IEEE Computer Society, 1999.

[43] S. Xu and C.-T. Huang. Attacks on PKM protocols of IEEE 802.16 and its later versions. In *Proceedings of 3rd International Symposium on Wireless Communication Systems (ISWCS 2006)*, Valencia, Spain, 2006.

[44] S. Xu, M.M. Matthews, and C.-T. Huang. Security issues in privacy and key management protocols of IEEE 802.16. In *Proceedings of the 44th ACM Southeast Conference (ACMSE 2006)*, Melbourne, Florida, USA, 2006.