

# Developing Security Protocols by Refinement

**Report****Author(s):**

[Sprenger, Christoph](#) ; [Basin, David](#) 

**Publication date:**

2013-06

**Permanent link:**

<https://doi.org/https://doi.org/10.3929/ethz-a-009900739>

**Rights / license:**

[In Copyright - Non-Commercial Use Permitted](#)

**Originally published in:**

D-INFK Technical Report 787

# Developing Security Protocols by Refinement

Christoph Sprenger and David Basin  
*Institute of Information Security, ETH Zurich*  
{sprenger,basin}@inf.ethz.ch

June 21, 2013

## Abstract

We propose a development method for security protocols based on stepwise refinement. Our refinement strategy transforms abstract security goals into protocols that are secure when operating over an insecure channel controlled by a Dolev-Yao-style intruder. As intermediate levels of abstraction, we employ message-less guard protocols and channel protocols communicating over channels with security properties. These abstractions provide insights on why the protocols are secure and foster the development of families of protocols sharing common structure and properties. We have implemented our method in Isabelle/HOL and used it to develop different entity authentication and key establishment protocols. These protocols include realistic features such as key confirmation, replay caches, and encrypted tickets. Our development highlights that guard protocols and channel protocols provide fundamental abstractions for bridging the gap between security properties and standard protocol descriptions based on cryptographic messages. It also shows that our refinement approach scales to protocols of nontrivial size and complexity.

**Keywords:** Security protocols, stepwise refinement, correct-by-construction development, entity authentication, key establishment.

## 1 Introduction

The fact that the development of even simple security protocols is error prone motivates the use of formal methods to ensure their security. Past decades have witnessed significant progress in post-hoc verification methods for protocol security based on model checking and theorem proving such as [47, 18, 6, 8, 26, 50]. However, methods for developing security protocols lag behind and protocol design remains more an art than a science.

In our view, a development method should be systematic and hierarchical. By this we mean that the development is decomposed into smaller steps that are easy to understand and that these steps should span well-defined abstraction levels leading the developer from requirements down to cryptographic protocols. Moreover, the resulting protocols should be secure in well-established attacker models and such claims should

level	name	features
L0	security properties	global, protocol-independent
L1	guard protocols	roles, local store, no messages, no intruder
L2	channel protocols	channels with security properties, intruder
L3	crypto protocols	cryptographic messages, Dolev-Yao intruder

Table 1: Levels of the refinement strategy

ideally be supported by machine-checked formal proofs. Stepwise refinement provides such a hierarchical development method. However, most existing refinement-based approaches to developing security protocols [22, 16, 35, 14, 24, 27] fall short of at least one of these desiderata.

In this paper, we advocate a development method based on stepwise refinement that satisfies all requirements on our wish list. Its central element is a four-level refinement strategy, introduced in Section 3 and summarized in Table 1. This strategy allows developers to build models that incrementally incorporate the system requirements and environment assumptions. Each model constitutes an idealized functionality for subsequent refinements. Safety properties, once proved for a model, are preserved by further refinements.

Level 0 of our refinement strategy consists of abstract, protocol-independent models of *security properties* such as (reachability-based) secrecy and authentication. At Level 1 of *guard protocols*, we introduce protocol roles, their local state, and the basic sequencing of the protocol steps. The protocol agents communicate by accessing each other’s local states. At Level 2 of *channel protocols*, the agents exchange plaintext messages over communication channels with intrinsic security properties, e.g., confidential or authentic channels. Finally, at Level 3 of *cryptographic protocols*, we replace these messages by cryptographic messages transmitted over insecure channels controlled by a standard Dolev-Yao intruder. In our method, the functional and security requirements are established at the first two levels, while the last two levels incorporate the environment assumptions (i.e., the hostile distributed environment).

A central and novel feature of our approach is the use of guard protocols (L1) as an intermediate abstraction linking security properties (L0) and message-based protocols (L2-3). Guard protocols enable the straightforward abstract realization of security goals by adding *security guards* as necessary conditions for the execution of certain protocol steps. Different kinds of security guards ensure the preservation of different properties such as secrecy, authentication, and recentness. For example, key secrecy means that only authorized agents may know a key. Accordingly, steps of guard protocols where an agent  $A$  learns a key  $K$  contain a guard requiring that  $A$  is authorized to know  $K$ . For authentication, there are guards ensuring that the local state of an agent (partially) agrees with the state of another agent.

The security guards for secrecy and authentication specify conditions on the global state in terms of other agent’s local states. This realizes an abstract form of communication. This abstraction simplifies proofs, but it is not directly implementable in a distributed setting. Hence, we implement these guards at Level 2 by receiving messages on channels with intrinsic security properties. The associated refinement proof

naturally gives rise to invariants stating that the receiving of channel messages implies the security guards they implement. These invariants precisely state the security properties guaranteed by the messages. For example, a message containing a key  $K$  received on a confidential channel to an agent  $A$  may implement a guard authorizing  $A$  to learn  $K$ . The corresponding invariant guarantees that  $A$  is authorized to learn  $K$  from this message.

To validate the effectiveness of our refinement strategy, we developed different authentication and key establishment protocols from abstract specifications. In Section 3, we develop two simple unilateral authentication protocols as running examples illustrating our approach: the ISO/IEC 9798-3 protocol and the first two steps of the Needham-Schroeder-Lowe protocol. In Section 4, we show how to systematically develop an entire family of key transport protocols. This family consists of the Needham-Schroeder Shared-key protocol, the core of Kerberos 4 and Kerberos 5, and the Denning-Sacco protocol. Compared to the running examples from Section 3, these protocols are significantly more complex in size and message structure and exhibit additional features and security properties such as the use of timestamps, replay protection caches, encrypted tickets (double encryption), dynamically created communication channels, key confirmation, key freshness, and key recentness. Interestingly, in each class of protocols, the final models refine a common Level 1 ancestor, even though they use different cryptographic primitives and communication patterns.

**Contributions** We see our main contributions as follows. Our first contribution is methodological. Our initial models at Level 0 specify the security goals of the guard protocols at Level 1. These in turn determine the basic structure of entire families of protocols. Using the refinement strategy outlined above, we *systematically* refine these abstract models in a series of well-defined steps to protocols using cryptographic messages sent over an insecure channel. Our refinement strategy aims at proving security properties at the highest possible abstraction level. General results guarantee that these properties are preserved by subsequent refinements.

Our refinement strategy naturally gives rise to straightforward parametrized simulation relations between the state spaces of models at different levels. These relations are instantiated and used in refinement proofs. Moreover, the process of proving refinements helps us discover invariants, many of which are canonical. For example, the simulation relation linking Levels 2 and 3 usually expresses that the local states of the roles is untouched by the refinement and maps the cryptographic messages at Level 3 to messages on abstract channels at Level 2. A canonical invariant that appears in such refinement proofs states that the honest agents' long-term keys remain secret. This is the natural level of abstraction for this invariant. Typically, the other relevant security properties are already proved in earlier refinements.

Second, our development provides evidence that guard protocols and channel protocols constitute two fundamental abstractions that bridge the gap between security properties and standard protocol descriptions based on cryptographic messages. In other approaches, the guarantees about protocol messages given by the invariants mentioned above and the associated reasoning are usually stated informally, if at all. By formalizing them, our approach fosters clear protocol designs and abstract, simple se-

curity proofs. Moreover, in standard protocol descriptions, secrecy and authentication are often not clearly separated (e.g., when using a secure channel providing both properties) or are interdependent (e.g., due to a layered use of cryptographic keys and operations). This appears to be a major source of complexity and errors and makes security protocols hard to design and understand. In contrast, guard protocols realize secrecy and authentication properties abstractly, independently, and straightforwardly. This facilitates the formal development of security protocols and underscores the central role that guard protocols can play in property-driven design approaches.

At the next level, channel protocols allow us to reason about a protocol’s security properties at a lower degree of complexity than with the Dolev-Yao intruder. The channels also enable a range of different realizations. For example, we may implement an authentic channel using signatures or MACs. Moreover, the communication structure may change from Level 2 to 3. For instance, an abstract key server may independently distribute keys on secure channels to the initiator and the responder, whereas in the concrete realization the distribution is sequential: one role receives two encrypted copies of the key and forwards one copy to the other role (Section 4). The abstract view represents the essential structure of server-based key distribution. The forwarding is just an implementation detail.

Third, we show how refinement can be used to develop protocols that are secure under the standard Dolev-Yao intruder model (at Level 3). In contrast, in related work such as [16, 35, 22, 14], the authors do not continue the refinements down to the level of a standard Dolev-Yao model based on an algebra of cryptographic messages; some use ad-hoc, protocol-dependent intruder definitions. This makes it difficult to compare their models with existing work on protocol modeling and verification and to know for which adversaries their properties hold.

Fourth, we show how to develop an entire family of key transport protocols from requirements down to protocols that are secure against a standard Dolev-Yao intruder. We model realistic features that are often abstracted away, such as replay prevention caches for timestamped messages to achieve strong properties like injective authentication. We have formalized all models and proofs in this paper in the Isabelle/HOL theorem prover.<sup>1</sup> Our formalization includes an infrastructure with general Isabelle/HOL theories for protocol runs, fresh values, and channels with security properties. Our development shows that our method scales to protocols of realistic complexity. The protocols in our development share both structure and properties as the graph of refinements of our development indicates (see Figure 5). Property preservation through refinements avoids proof duplication and fosters well-structured proofs.

Our final contribution is a comprehensive definitional extension of Isabelle/HOL with a theory of refinement that is based on simulation and borrows elements from [4, 2]. We define an implementation relation on models including a notion of observation, derive proof rules for invariants and refinement, and show that refinement is a sound method for establishing the implementation relation between models.

---

<sup>1</sup>Our entire development is available online at <http://people.inf.ethz.ch/cspreng/SecProtoRefine>.

**Organization** The remainder of this paper is organized as follows. In Section 2, we introduce Isabelle/HOL, notational conventions, and summarize the theory of refinement that we have embedded in Isabelle/HOL. In Section 3, we present our four-level refinement strategy for security protocols and illustrate its application by deriving two simple authentication protocols. In Section 4, we report on our development of a family of key transport protocols. In Section 5, we further discuss related work and we draw conclusions in Section 6.

## 2 Preliminaries

### 2.1 Isabelle/HOL and notation

Isabelle is a generic, tactic-based theorem prover. We have used Isabelle’s implementation of higher-order logic, Isabelle/HOL [46], for our developments. HOL can roughly be seen as logic on top of a functional programming language. We assume that the reader has basic familiarity with both logic and typed functional programming. Proof automation in Isabelle is supported by a rewrite-based simplifier and a tableau procedure. These are invoked in isolation or combined using proof tactics.

To enhance readability, we will use standard mathematical notation where possible and blur the distinction between types and sets. We also drop typing information unless it is essential to understand a definition.

We use two definitional equalities:  $\equiv$  for terms and  $\triangleq$  for types. We define partial functions by  $A \mapsto B \triangleq A \rightarrow B_{\perp}$ , where  $B_{\perp} \triangleq B \mid \perp$ .<sup>2</sup> The term  $f(x \mapsto y)$  denotes the function that behaves like  $f$ , except that it maps  $x$  to  $y$ . For a function or binary relation  $R \subseteq A \times B$  and a set  $X \subseteq A$ , we define the image of  $X$  under  $R$  by  $R(X) \equiv \{y \in B \mid \exists x \in X. (x, y) \in R\}$ . The inductive type of lists is defined by  $[A] \triangleq [] \mid A\#[A]$ , where  $[]$  is the empty list and  $a\#l$  is the list built by prefixing the element  $a \in A$  to the list  $l \in [A]$ . We write  $[1, 2, 3]$  for  $1\#2\#3\#[]$ . We define multisets over  $A$  by  $\text{multiset}(A) \triangleq A \rightarrow \mathbb{N}$ . For multisets  $m, n \in \text{multiset}(A)$ , the term  $m(e)$  indicates the multiplicity of  $e$  in  $m$  and union is defined by  $(m \uplus n)(e) = m(e) + n(e)$ . Record types may be defined, such as  $\text{point} \triangleq \langle x \in \mathbb{N}, y \in \mathbb{N} \rangle$  with elements like  $r = \langle x = 1, y = 2 \rangle$  and projections  $r.x$  and  $r.y$ . The term  $r(x := 3)$  denotes  $r$ , where  $x$  is updated to 3, i.e.,  $\langle x = 3, y = 2 \rangle$ . The type  $\text{cpoint} \triangleq \text{point} + \langle c \in \text{color} \rangle$  extends  $\text{point}$  with a  $\text{color}$  field. For record types  $T$  and  $U$  including fields  $F$ , we define  $\Pi_F^{T,U} \equiv \{(r, s) \in T \times U \mid \bigwedge_{x \in F} r.x = s.x\}$ . If  $U$  has exactly the fields  $F$ , the function  $\pi_F^{T,U} : T \rightarrow U$  projects  $T$  to  $U$ . We will drop the superscripts  $T$  and  $U$  from  $\Pi_F^{T,U}$  and  $\pi_F^{T,U}$  when they are clear from the context.

### 2.2 Refinement theory

A development by refinement starts from a set of system requirements and environment assumptions. We then construct a series of models resulting in a system that fulfills the requirements provided it runs in an environment satisfying the assumptions. We

<sup>2</sup>More precisely,  $B_{\perp} \triangleq \text{Some}(B) \mid \perp$ , but we will often elide single-argument constructors.

summarize the refinement theory that ‘we developed in Isabelle/HOL. It is inspired by [3, 2].

### 2.2.1 Specification and implementation

We define the structure of our models and we formalize the meaning of implementation.

**Definition 2.1.** A *specification* is a triple of the form  $S = (T, O, obs)$ , where  $T = (\Sigma, \Sigma_0, \rightarrow)$  is a *transition system* with state space  $\Sigma$ , set of initial states  $\Sigma_0 \subseteq \Sigma$ , and transition relation  $\rightarrow \subseteq \Sigma \times \Sigma$ . The *observation function*  $obs : \Sigma \rightarrow O$  maps states to elements of a set of observations  $O$ .

The definition of transition systems is standard. The observation function  $obs$  specifies which state information is visible to an outside observer.

We will work with structured specifications, where the state space  $\Sigma$  is a record type, i.e., a set of tuples of state variables, and the transition relation  $\rightarrow$  is a finite union of parametrized relations, called *events*. Events have the form

$$evt(\bar{x}) = \{(s, s') \mid G(\bar{x}, s) \wedge s'.\bar{v} := \bar{f}(\bar{x}, s)\},$$

where  $\bar{\cdot}$  denotes vectors. Here,  $\bar{x}$  are the parameters,  $\bar{v}$  are the state variables,  $G(\bar{x}, s)$  is a conjunction of *guards*, and  $s'.\bar{v} := \bar{f}(\bar{x}, s)$  is an *action* with *update functions*  $\bar{f}$ . The guards depend on the parameters  $\bar{x}$  and the current state  $s$  and determine when the event is enabled. The action is syntactic sugar denoting the relation  $s' = s(\bar{v} := \bar{f}(\bar{x}, s))$ , i.e., the simultaneous assignment of values  $\bar{f}(\bar{x}, s)$  to the variables  $\bar{v}$  in the state  $s$ , yielding the state  $s'$ .

**Example 2.2.** Consider an abstract file transfer protocol specification

$$S_f \equiv ((\Sigma_f, \Sigma_f, \rightarrow_f), O_f, id),$$

where  $\Sigma_f \triangleq \langle f, g \in file \rangle$  with  $file \triangleq I \rightarrow D$  for a finite index set  $I$  and a set of data blocks  $D$  and  $\rightarrow_f \equiv xfer_f$ . The event  $xfer_f \equiv \{(s, s') \mid s'.g := s.f\}$  transfers the file  $f$  in one shot to  $g$ . All states are possible initial states and the entire state is observable, i.e.,  $O_f = \Sigma_f$  and the observation function is the identity.

The set of behaviors of  $S$ ,  $beh(S)$ , consists of the finite sequences  $s_0 \cdots s_n$  of states that start in an initial state in  $s_0 \in \Sigma_0$  and are linked by transitions in  $\rightarrow$ , that is,  $s_i \rightarrow s_{i+1}$  for  $0 \leq i < n$ . The set  $reach(S)$  denotes the set reachable states. Since we only consider safety properties (in fact, invariants), it suffices to consider just finite behaviors. The sets of *observable behaviors* and *reachable observations* of the specification  $S$  are defined by  $obeh(S) \equiv obs(beh(S))$  and  $oreach(S) \equiv obs(reach(S))$ , where  $obs$  is applied to behaviors elementwise.

Our notion of one specification implementing another is defined by the inclusion of their observable behaviors.

**Definition 2.3.** We say  $S_c$  implements  $S_a$  via the mediator function  $\pi : O_c \rightarrow O_a$  if  $\pi[obeh(S_c)] \subseteq obeh(S_a)$ .



Figure 1: Refinement of events (left) and observations (right)

We consider two types of invariants: internal ones are supersets of  $reach(S)$  and external ones are supersets of  $oreach(S) \equiv obs(reach(S))$ . We use internal invariants to strengthen simulation relations in refinement proofs (see Example 2.9 below). The importance of external invariants is that they are preserved by implementations.

**Proposition 2.4 (Invariant preservation).** *Suppose  $S_c$  implements  $S_a$  via the mediator function  $\pi$  and  $oreach(S_a) \subseteq J$  for some  $J \subseteq O_a$ . Then  $\pi(oreach(S_c)) \subseteq J$ .*

The observation and mediator functions allow us to relate specifications with differently structured state spaces. In particular, the mediator function enables the addition of details to observations during implementation. Proposition 2.4 guarantees a well-defined notion of property preservation for series of implementations. We use refinement as a proof method to establish implementations.

## 2.2.2 Refinement

Our notion of refinement is based on standard simulation [40], which we extend to account for observations. Figure 1 illustrates the second and third point of the following definition.

**Definition 2.5.** We say  $S_c = (T_c, O_c, obs_c)$  *refines*  $S_a = (T_a, O_a, obs_a)$  using the simulation relation  $R \subseteq \Sigma_a \times \Sigma_c$  and the mediator function  $\pi : O_c \rightarrow O_a$ , written  $S_c \sqsubseteq_{R, \pi} S_a$ , if the following three conditions are met.

1. Each concrete initial state is related to some abstract initial state, i.e.,

$$\Sigma_{c,0} \subseteq R(\Sigma_{a,0}).$$

2. For each concrete event  $evt_c(\bar{x})$  there is an abstract event  $evt_a(\bar{z})$  simulating it, i.e.,

$$R; evt_c(\bar{x}) \subseteq evt_a(\bar{w}(\bar{x})); R,$$

where ‘;’ is forward relational composition and  $\bar{w}(\bar{x})$  are the *witnesses* that construct parameters for  $evt_a$  from those of  $evt_c$ .

3.  $R$  respects observations mediated by  $\pi$ , i.e., for all  $(s, t) \in R$ ,

$$obs_a(s) = \pi(obs_c(t)).$$

We say  $S_c$  *refines*  $S_a$  using  $\pi$ , written  $S_c \sqsubseteq_{\pi} S_a$ , if  $S_c \sqsubseteq_{R, \pi} S_a$  for some  $R$ .

Let the concrete and abstract system have state variables  $\bar{v}$  and  $\bar{u}$ , respectively. Moreover, suppose the event  $evt_c(\bar{x})$  has guards  $G_c$  and update functions  $\bar{f}_c$  and the abstract event  $evt_a(\bar{z})$  has guards  $G_a$  and update functions  $\bar{f}_a$ . Condition 2 decomposes into two proof obligations, called *guard strengthening* and *action refinement*, both under the premises  $(s, t) \in R$  and  $G_c(\bar{x}, t)$ .

$$\bullet G_a(\bar{w}(\bar{x}), s) \quad (\mathbf{GRD})$$

$$\bullet (s(\bar{u} := \bar{f}_a(\bar{w}(\bar{x}), s)), t(\bar{v} := \bar{f}_c(\bar{x}, t))) \in R \quad (\mathbf{ACT})$$

Guard strengthening requires that if the concrete event is enabled then so is the abstract one. Action refinement expresses that the two states resulting from the execution of the abstract and concrete actions are again related by  $R$ .

We assume that all models include a special event *skip* (the identity relation), used to simulate new concrete events. The intuition is that events added to the concrete model correspond to an unobservable stuttering step in the abstract model.

Refinement is reflexive and transitive. Moreover, refinement is a sound (but incomplete [2, 36]) method to establish implementation.

**Proposition 2.6 (Pre-order).** *We have (i)  $S \sqsubseteq_{id} S$  and (ii)  $S_3 \sqsubseteq_{\rho} S_2$  and  $S_2 \sqsubseteq_{\pi} S_1$  imply  $S_3 \sqsubseteq_{\pi \circ \rho} S_1$ .*

**Proposition 2.7 (Soundness of refinement).** *If  $S_c \sqsubseteq_{\pi} S_a$  then  $S_c$  implements  $S_a$  via the mediator function  $\pi$ .*

The combination of Propositions 2.4, 2.6, and 2.7 ensures that requirements and assumptions, once established as external invariants, are preserved by subsequent refinements. Note that this does not generally hold for internal invariants. For later reference we state the following corollary.

**Corollary 2.8 (Invariant preservation II).** *If  $S_c \sqsubseteq_{\pi} S_a$  and then  $oreach(S_a) \subseteq J$  for some  $J \subseteq O_a$ . Then  $\pi(oreach(S_c)) \subseteq J$ .*

**Example 2.9.** We define a “protocol” implementing the file transfer specification  $S_f$  by  $S_p \equiv ((\Sigma_p, \Sigma_{p,0}, \rightarrow_p), O_p, obs_p)$ , where  $\Sigma_p \triangleq \Sigma_f + (\mid b \in I \rightarrow D \mid)$  extends the state  $\Sigma_f$  with a buffer  $b$ . The set  $\Sigma_{p,0}$  consists of initial states of the form  $(\mid f = f_0, g = g_0, b = \emptyset \mid)$  for some  $f_0, g_0 \in file$  and the empty buffer  $b$ . The protocol non-deterministically transfers blocks of the file  $f$  into the buffer  $b$  from where it is assigned to  $g$ , once the transfer is completed. The transition relation  $\rightarrow_p \equiv xfer_p \cup \bigcup_{i \in I} blk_p(i)$  is the union of two events:

$$blk_p(i) \equiv \{(t, t') \mid i \in I \setminus dom(t.b) \wedge t'.b := t.b(i \mapsto f(i))\}$$

and  $xfer_p \equiv \{(t, t') \mid dom(b) = I \wedge t'.g := t.b\}$ . The observation function  $obs_p \equiv \pi_f$  projects the state  $\Sigma_p$  to the observations  $O_p = \Sigma_f$ .

Let us try to establish a refinement between  $S_p$  and  $S_f$ , using the simulation relation  $R \equiv \Pi_f \subseteq S_f \times S_p$ , i.e., the inverse of the projection  $\pi_f : \Sigma_p \rightarrow \Sigma_f$ , and the identity mediator function  $\pi \equiv id$ . We focus on point (2), where we must show that  $blk_p(i)$  refines *skip* and that  $xfer_p$  refines  $xfer_f$ . The guard strengthening (**GRD**) proof obligation is trivial in both cases, since the abstract guards are true. The action refinement

(ACT) proof obligation for  $blk_p(i)$  and  $skip$  (the identity relation) requires showing  $(s, t') \in \Pi_f$  for  $t' = t \langle b := t.b(i \mapsto f(i)) \rangle$ , assuming  $(s, t) \in \Pi_f$  and  $i \in I$ . This holds trivially, since  $t'.g = t.g = s.g$ . In the action refinement for  $xfer_p$  and  $xfer_f$ , we must show that  $(s \langle g := s.f \rangle, t \langle g := t.b \rangle) \in \Pi_f$  assuming  $(s, t) \in \Pi_f$  and  $dom(b) = I$ . To prove this, we need additional information about the relation between  $b$  and  $f$ , expressed as the internal invariant  $I_p \equiv \{t \in \Sigma_p \mid \forall i \in dom(b). t.b(i) = t.f(i)\}$  of  $S_p$ . We establish this invariant separately and use it to strengthen the simulation relation to  $R \equiv \Pi_f \cap (\Sigma_f \times I_p)$ .

In further refinements, one could develop a more realistic implementation, for example, by eliminating non-determinism and by modeling a communication medium such as an unreliable channel with acknowledgement messages.

### 3 Security Protocol Refinement

We now present our framework for security protocol development by refinement. Each development starts by defining the system requirements and the environment assumptions. The environment assumptions include the adversary model and the cryptographic setup. Given these elements we need a *refinement strategy* telling us in which order to incorporate them into our models. The crucial point is that requirements and assumptions, once modeled, are *preserved* by subsequent refinement steps (Corollary 2.8).

The following four-level refinement strategy guides our developments, where each level may itself consist of several refinement steps.

**Level 0** *Security properties.* We give abstract, protocol-independent specifications of secrecy and authentication properties. The models' states contain just enough structure to formulate these properties as invariants and define a few abstract event satisfying these invariants. There are no intruder events.

**Level 1** *Guard protocols.* These are abstract protocols without message passing. We introduce protocol roles, local states of agents, and basic protocol steps. Agents read data directly from other agents' local states. There are still no intruder events.

**Level 2** *Channel protocols* communicate over abstract channels with security properties, such as confidential and authentic channels. The intruder may eavesdrop messages on non-confidential channels and fake messages on non-authentic channels. No cryptography is used.

**Level 3** *Cryptographic protocols.* The messages on the abstract channels from Level 2 are now implemented using cryptographic messages sent over insecure channels. A standard Dolev-Yao intruder completely controls the network.

In the rest of this section, we introduce protocol-independent infrastructure for modeling and reasoning about protocol runs, fresh values, channels with security properties, and intruder behavior. We also describe for each level the relevant structures such as type definitions, state variables, and simulation relations. As running examples, we develop two simple authentication protocols. In Section 4, we will apply this framework

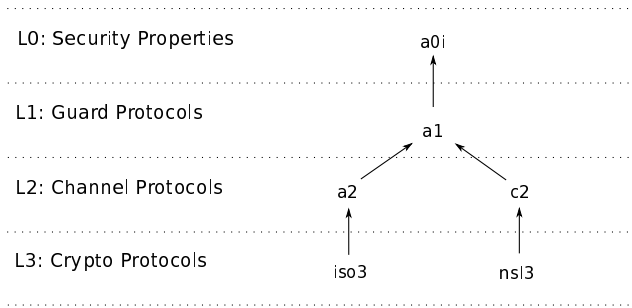


Figure 2: Refinement graph for authentication protocols

to a more complex development and construct a family of key establishment protocols. We start by introducing our running example.

### 3.1 Entity authentication protocols

As an illustration of our methodology, we develop two unilateral authentication protocols. Both protocols are based on a standard challenge-response pattern, where the initiator sends a nonce as a challenge to the responder who returns it in a cryptographically transformed form that authenticates him to the initiator. The first protocol is the signature-based two-pass ISO/IEC 9798-3 standard [31]. The second, which we call NSL/2, consists of the first two steps of the Needham-Schroeder-Lowe protocol [33] and uses public-key encryption.

We start by specifying the system requirements and by making our assumptions about the environment explicit. Our notion of entity authentication is based on the strong property of injective agreement [34]. A protocol satisfies this property, if given two roles  $R$  and  $R'$  and a set of data items  $d$  then for each protocol thread executing role  $R$  there exists exactly one thread executing role  $R'$  with whom it agrees on the data items  $d$ .

**Requirement R1 (Protocol roles).** The protocol has two roles, which we call initiator and responder.

**Requirement R2 (Entity authentication).** The initiator injectively agrees with the responder on the initiator's nonce and possibly on additional data.

We assume a standard Dolev-Yao intruder that we identify as usual with the communication network. Moreover, we make two assumptions about agent corruption and the cryptographic setup.

**Assumption A1 (Dolev-Yao intruder).** The intruder controls the network. He receives all messages sent and he can build and send messages composed from parts obtained by decomposing received messages using the cryptographic keys he knows.

**Assumption A2 (Static corruption).** An arbitrary fixed subset of agents is corrupted, whereby their long-term keys are exposed to the intruder.

**Assumption A3 (Cryptographic setup).** The requisite cryptographic keys are distributed prior to protocol execution.

**Development overview** Figure 2 summarizes the refinements in our development in a *refinement graph*. Each node represents a model and each arc  $m \rightarrow m'$  represents a refinement  $m \sqsubseteq_{\pi} m'$  for some mediator function  $\pi$ , not shown.

We progress from the initial model  $a0i$  representing the intended injective agreement property at Level 0 (Section 3.3) to the guard protocol  $a1$  at Level 1 (Section 3.4). This guard protocol is refined into two channel protocols,  $a2$  and  $c2$ , based on authentic and confidential channels, respectively (Section 3.5). Finally, we realize these two protocols as cryptographic protocols,  $iso3$  and  $ns13$  (Section 3.6). In the following, we will restrict our presentation to the left path in the refinement graph. We do not present the models  $c2$  and  $ns13$ , whose development is similar to  $a2$  and  $iso3$ 's.

## 3.2 General setup

We describe our formalization of atomic messages: agents, keys, nonces, and numbers. We start by defining a type of agents, *agent*. We assume a non-empty subset *bad* of dishonest agents (with complement *good*), an honest server  $S \in \text{good}$ , and an intruder  $i \in \text{bad}$  with access to all dishonest agents' long-term keys.

We also need a mechanism to generate fresh nonces and keys. We assume a type *rid* of identifiers that we will use to uniquely identify protocol runs at Levels 1 to 3. From this type, we derive the type of freshness identifiers as  $\text{fresh} \triangleq \text{mkf}(\text{rid}, \mathbb{N})$ , which has a single constructor *mkf*. We write  $\text{mkf}(R, i)$  as  $R\$i$ . This setup allows us to derive an arbitrary number of unique freshness identifiers from each protocol run identifier. We define the types of nonces and keys as follows.

$$\text{nonce} \triangleq \text{fresh} \quad \text{key} \triangleq \text{fresh} \mid \text{ltk}$$

Nonces and session keys both use freshness identifiers. The type of long-term keys, *ltk*, is left unspecified at this point.

Finally, we define the type *atom* of atomic messages as the disjoint sum of the types of agents, nonces, keys, and numbers:

$$\text{atom} \triangleq \text{agent} \mid \text{key} \mid \mathbb{N}.$$

We use numbers mainly as timestamps. We will usually omit constructors from atomic messages and use a notational convention instead. In particular, we use  $A, B, C$  for agents,  $N, Na, Nb$  for nonces,  $K, Kab$  for session keys, and  $T, Ta, Ts$  for timestamps.

Many protocols assume a setup of long-term keys, which is established out-of-band before the protocol starts. We model this by assuming an abstract (uninterpreted) key setup  $\text{keySetup} \subseteq \text{key} \times \text{agent}$  defining the initial key knowledge of each agent. The definitions of this relation and the type *ltk* are deferred to Level 3 (Section 3.6), since they are protocol-dependent. For example, the protocol may use a PKI or a shared-key

setup. The set of statically corrupted keys is derived from the key setup as the keys initially known by dishonest agents:

$$\text{corrKey} \triangleq \text{keySetup}^{-1}(\text{bad}).$$

### 3.3 Level 0 — Security properties

We present abstract, protocol-independent models of secrecy and authentication and we formalize and prove their relevant properties as external invariants. Each protocol development starts with the formalization of the protocol's security requirements. This is achieved by appropriately instantiating these Level 0 models. We will later show that our guard protocol models at Level 1 refine these instantiated models, thus establishing the respective requirements (by Corollary 2.8).

#### 3.3.1 Secrecy

Our first model abstractly captures the notion of secrecy. We introduce two state variables,  $kn$  and  $az$ , both relations between data (of polymorphic type  $\delta$ ) and agents. In particular,  $(d, A) \in s.kn$  means that agent  $A$  knows data  $d$  in state  $s$ , and  $(d, A) \in s.az$  means that agent  $A$  is authorized to know data  $d$  in state  $s$ . The entire state is observable.

$$\begin{aligned} \Sigma_{s0}(\delta) \triangleq (& \langle kn \in \mathcal{P}(\delta \times \text{agent}), & \text{-- knowledge relation} \\ & az \in \mathcal{P}(\delta \times \text{agent}) \rangle & \text{-- authorization relation} \end{aligned}$$

Secrecy can be expressed as the property that all knowledge is authorized.

$$\text{secrecy} \equiv \{s \mid s.kn \subseteq s.az\}$$

In other words, agents know secrets only if they are authorized. We allow any state satisfying this property as an initial state.

**Events** The model  $s0$  has one event for secret generation and one for learning secrets. The secret generation event is parametrized by the data  $d$ , an agent  $A$ , and the intended group of agents  $G$  sharing  $d$ .

$$\begin{aligned} \text{gen}_{s0}(d, A, G) \equiv & \{(s, s') \mid \\ & \text{-- guards} \\ & d \notin \text{dom}(s.kn) \wedge & \text{-- } d \text{ is fresh} \\ & A \in G \wedge & \text{-- } A \text{ is a member of } G \\ & \text{-- actions} \\ & s'.kn := s.kn \cup \{(d, A)\} \wedge \\ & s'.az := s.az \cup \{(d, B) \mid B \in G \vee G \cap \text{bad} \neq \emptyset\} \} \end{aligned}$$

The guards require that  $d$  is fresh, i.e., not known to any agent, and that  $A$  belongs to the group  $G$ . The first action adds the pair  $(d, A)$  to the knowledge relation  $s.kn$ . The second action updates the authorization relation with  $\{d\} \times G$  if all agents in  $G$  are honest and with  $\{d\} \times \text{agent}$  otherwise. That is, if the group  $G$  contains a dishonest member, there is no point in restricting access to  $d$ .

In the secret-learning event, an authorized agent  $B$  learns the secret  $d$ .

```

learns0(d, B) ≡ { (s, s') |
  -- guards
  (d, B) ∈ s.az ∧           -- B is authorized to know d
  -- action
  s'.kn := s.kn ∪ {(d, B)} }

```

From a secrecy perspective, it is irrelevant from whom  $B$  learns  $d$ . Authentication aspects will be covered separately. The model  $s0$  clearly preserves secrecy.

**Proposition 3.1.** *Let  $s0$  be the above specification. Then  $\text{oreach}(s0) \subseteq \text{secrecy}$ .*

### 3.3.2 Authentication

Our notion of authentication is based on Lowe's agreement [34]. Informally, a role  $R$  *non-injectively agrees* with the role  $R'$  on the data  $d$  if whenever an honest agent  $A$  in role  $R$  terminates a run, apparently with an honest agent  $B$  in role  $R'$ , then there is a run of agent  $B$  in role  $R'$  with whom he agrees on the participants, their roles, and the data  $d$ . This agreement is *injective* if there is at most one run of role  $R$  that agrees with a given run in role  $R'$ .

We formulate two models,  $a0n$  and  $a0i$ , that represent a minimal, extensional variant of non-injective and injective agreement using signals, which indicate particular stages of each role's progress (e.g., termination). The state record has as its single field an initially empty multiset of signals,  $\text{sigs}$ . The entire state is observable.

$$\begin{aligned} \text{signal}(\delta) &\triangleq \text{Running}(\text{list}(\text{agent}) \times \delta) \mid \text{Commit}(\text{list}(\text{agent}) \times \delta) \\ \Sigma_{a0}(\delta) &\triangleq \langle \text{sigs} \in \text{multiset}(\text{signal}(\delta)) \rangle \end{aligned}$$

There are two signals:  $\text{Running}(h, d)$  and  $\text{Commit}(h, d)$ , where  $h$  is a list of agents and  $d$  is data of polymorphic type  $\delta$  that is instantiated later. The agreement on the data  $d$  is, by convention, between the first two agents in  $h$  and assumes the honesty of all agents in  $h$ . In the simplest case,  $h$  includes the two agents participating in the agreement, but in some cases it is necessary to include other agents in  $h$ , such as those agents relaying messages.

Non-injective agreement states that if the agents in  $h$  are honest and there is a  $\text{Commit}(h, d)$  signal (thought to be raised by the first agent in  $h$ ), then there is a matching  $\text{Running}(h, d)$  signal (raised by the second agent in  $h$ ).

$$\begin{aligned} \text{niagree}_{a0n} &\equiv \{s \mid \forall h, d. \\ &h \subseteq \text{good} \wedge s.\text{sigs}(\text{Commit}(h, d)) > 0 \rightarrow s.\text{sigs}(\text{Running}(h, d)) > 0 \} \end{aligned}$$

Injective agreement strengthens this by requiring that the number of  $\text{Commit}(h, d)$  signals is not greater than the number of matching  $\text{Running}(h, d)$  signals.

$$\text{iagree}_{a0i} \equiv \{s \mid \forall h, d. h \subseteq \text{good} \rightarrow s.\text{sigs}(\text{Commit}(h, d)) \leq s.\text{sigs}(\text{Running}(h, d)) \}$$

The models  $a0n$  and  $a0i$  have two events,  $\text{running}(h, d)$  and  $\text{commit}(h, d)$ , which add the corresponding signal to the multiset  $s.\text{sigs}$ . The first event adds a  $\text{Running}(h, d)$  signal to the multiset  $s.\text{sigs}$ .

$$running_{a0n}(h, d) \equiv \{(s, s') \mid s'.sigs := s.sigs \uplus \{\text{Running}(h, d)\}\}$$

The second event adds a  $\text{Commit}(h, d)$  signal to the multiset  $s.sigs$ . Its guard requires that there is a matching  $\text{Running}(h, d)$  signal if the agents in  $h$  are honest. This ensures that invariant  $niagree_{a0n}$  is preserved.

$$\begin{aligned} a0n\_commit(h, d) &\equiv \{(s, s') \mid \\ &h \subseteq good \rightarrow s.sigs(\text{Running}(h, d)) > 0 \wedge \\ &s'.sigs := s.sigs \uplus \{\text{Commit}(h, d)\}\} \end{aligned}$$

The honesty condition weakens the guard just enough to accommodate the protocol's interaction with an explicit intruder in later refinements.

The model  $a0i$  for injective agreement is the same except for the guard in the event  $commit_{a0i}(h, d)$ , which we strengthen as follows to preserve the stronger invariant  $iagree_{a0i}$ .

$$h \subseteq good \rightarrow s.sigs(\text{Commit}(h, d)) < s.sigs(\text{Running}(h, d))$$

It is easy to see that  $a0i$  refines  $a0n$ . The properties of the models  $a0n$  and  $a0i$  are summarized as follows.

**Proposition 3.2.** *The models  $a0n$  and  $a0i$  defined above have the following properties: (i)  $oreach(a0n) \subseteq niagree_{a0n}$ , (ii)  $oreach(a0i) \subseteq iagree_{a0i}$ , and (iii)  $a0i \sqsubseteq_{Id, id} a0n$ .*

Since the variable  $sigs$  is observable, these invariants are preserved by further refinements (Corollary 2.8).

**Example 3.3 (Formalization of authentication properties).** Our example development starts with a formalization of the main security property: entity authentication by injective agreement (**R2**). We do this by instantiating the model  $a0i$ . Since we would like our entity authentication protocols to achieve injective agreement between the initiator  $A$  and the responder  $B$  on a nonce  $Na$  generated by  $A$  and also on a nonce  $Nb$  generated by  $B$ , we instantiate the type of data  $\delta$  in the model  $a0i$  to  $nonce \times nonce$ .

This model is very abstract. Further refinement steps are needed to obtain a protocol that is executable in the intended distributed environment and secure against a Dolev-Yao intruder as described by the environment assumptions. ♠

### 3.3.3 Other types of security property specifications

In contrast to establishing security properties by refinements of L0 models, we may express (and prove) security properties directly at the protocol level (i.e., at Level 1, 2, or 3) in one of two ways.

First, the property may be ensured by a *guard*, whereby the property is established by construction so that no extra proof is required. An example is a guard checking the validity of a timestamp to ensure the recentness of an associated session key. Second, we may formulate and prove the property as an *external invariant* of the protocol. This is how we express the freshness of session keys. An advantage of using the refinement of abstract models over these two alternatives is that the abstract models

are protocol-independent. This enables clear and uniform property specifications. In contrast, invariants that are formulated at the protocol level (see, e.g., [47]) must be specified individually and tend to be more complex.

### 3.4 Level 1 — Guard protocols

We now introduce protocol roles and runs. A run is a thread executed by some agent in a given role. Each run has a local memory holding state information. At this abstract level, runs share information by reading each other’s memory. We call such protocols *guard protocols*. Of course, this kind of communication by reading another thread’s memory is unrealistic in a distributed setting. Hence, at Level 2, we will refine this abstract form of communication by passing messages over communication channels.

#### 3.4.1 State

Guard protocols have at least one state variable, *runs*, which is a partial function mapping run identifiers (of type *rid*) to a run’s *local store*. This local store consists of the executed role, a pair of protocol participants, and a *frame* recording role-specific information. Our setup is designed for two-party protocols with an initiator (the first agent), a responder (the second agent), and possibly an additional fixed server S. This could easily be generalized to handle an arbitrary number of roles. The frame is a list of atomic messages that the run acquires during its execution.

$$\begin{aligned}
 \textit{role} &\triangleq \text{Init} \mid \text{Resp} \mid \text{Serv} \\
 \textit{frame} &\triangleq [\textit{atom}] \\
 \textit{runsT} &\triangleq \textit{rid} \rightarrow \textit{role} \times \textit{agent} \times \textit{agent} \times \textit{frame} \\
 \Sigma_1 &\triangleq (\mid \textit{runs} \in \textit{runsT} \mid)
 \end{aligned}$$

Here, we have schematically defined the state of a Level 1 protocol by the record type  $\Sigma_1$ . In concrete models, this state may contain additional variables. We assume that (at least) the variable *runs* is observable. All later refinements inherit the variable *runs*, but may add atoms to the run’s frames.

#### 3.4.2 Events

Each event executes a protocol step of a run by an agent in a particular role. The sequencing of events within a role is determined by *local guards* reading a run’s local store. For example, the guard  $\textit{runs}(R) = (\text{Init}, A, B, [Nb])$  expresses that the event executes a step of the run *R*, which is owned by the agent *A* playing the initiator role, talks to the responder *B*, and stores the nonce *Nb* in its frame. An event’s action typically extends the run frame with additional atomic messages, thereby tracking the run’s progress. We call a run *completed* if there is no event that extends its frame.

**Example 3.4 (Abstract authentication protocol, part I).** The state  $\Sigma_{a1}$  of our model *a1* contains a single variable, *runs*, defined above, which models threads executing protocol roles. Since our setup provides initiator and responder roles, this already

establishes Requirement **R1** that the protocol has two roles. Each role will generate a nonce that the other role records.

Let  $na$  and  $nb$  be arbitrary natural numbers. The three events of our specification  $aI$  abstractly model a protocol that follows a standard challenge-response pattern. The first event, which refines *skip*, just creates an initiator run  $Ra$  with an empty frame. The event also “generates” a nonce  $Na = Ra\$na$  associated with this run. Since  $Na$  is derived from the run identifier  $Ra$ , there is no need to record it in the frame.

```

step1aI(Ra,A,B,Na) ≡ {(s,s') |      -- by A, refines skip
  -- guards:
  Ra ∉ dom(s.runs) ∧                -- fresh run id
  Na = Ra$na ∧                       -- fresh nonce

  -- actions:
  s'.runs := s.runs(Ra ↦ (Init,A,B,[])) }
```

The second event refines *running<sub>a0i</sub>* and creates a responder run identified by  $Rb$  and the nonce  $Nb$ . The run acquires an *arbitrary* nonce  $Na$ , which need not come from an initiator, and records it in its frame. This reflects that the intruder can fake the challenge nonce in later refinements.

```

step2aI(Rb,A,B,Na,Nb) ≡ {(s,s') |  -- by B, refines runninga0i
  -- guards:
  Rb ∉ dom(s.runs) ∧                -- fresh run id
  Nb = Rb$nb ∧                       -- fresh nonce

  -- actions:
  s'.runs := s.runs(Rb ↦ (Resp,A,B,[Na])) }
```

The final event of the model  $aI$  will be presented in the next example. ♠

Agents communicate by reading their peers’ memories. This is achieved by *non-local guards* that refer to another run’s store. Such guards read remote values that may be compared with local values and used in local state updates. We have two kinds of non-local guards: *authorization guards* for secrecy and *authentication guards* for agreements. Authorization guards prevent unauthorized agents from learning secrets. We will explain the shape of these guards below.

An authentication guard for a given list of agents  $h$  and data  $d$  (cf. Section 3.3) executed by a run  $R$  requires the existence of a run  $R'$  executing a different role from  $R$ ’s and agreeing with  $R$  on data  $d$ , provided the agents in  $h$  are honest. More generally, for  $A$  in role  $r$  to agree with  $B$  in role  $r'$  on data  $d$  assuming honest agents  $h = [A, B, \dots]$ , we add an authentication guard  $G$  to an appropriate event of agent  $A$  (for example, the final event of its role). This guard requires that the agent  $B$  in role  $r'$  knows the data  $d$ , whenever the agents in  $h$  are honest. It has the form

$$G = h \subseteq \text{good} \rightarrow \exists R, \bar{x}. C(R, d) \wedge s.\text{runs}(R) = (r', A', B', \ell),$$

where  $R$  is the identifier of a run of  $B$  in role  $r'$ ,  $A'$  and  $B'$  are agent names possibly including  $A$  and  $B$ ,  $\ell$  is a list of atomic messages including those in  $d$ , and  $C(R, d)$

is a conjunction of equations fixing nonces and session keys in  $d$  generated by the run  $R$ . The free variables of  $G$  are exactly the variables appearing in  $h$  and  $d$ . All other variables are bound by the existential quantification over  $\bar{x}$ . We will see a concrete example shortly.

Since authorization and authentication guards are related to security properties, we also call them *security guards*. There are also local security guards, for example, which check the validity of timestamps to achieve recentness.

At this level of abstraction, there is no active intruder and guard protocol models therefore have no intruder events. Instead, the intruder is implicitly present in the event guards in the form of honesty assumptions.

**Example 3.5 (Abstract authentication protocol, part II).** The third step refines  $commit_{a0i}$  and models the initiator run  $Na$  receiving its nonce back from a responder run  $Nb$ . The first two guards state that the run  $Ra$  has not yet received the responder  $B$ 's nonce and has generated  $Na$ . The third guard is an authentication guard that ensures an agreement with the responder on the pair of nonces  $(Na, Nb)$ .

```

step3a1(Ra, A, B, Na, Nb) ≡ { (s, s') | -- by A, refines commita0i
  -- guards:
  s.runs(Ra) = (Init, A, B, []) ∧
  Na = Ra$na ∧
  (A ∉ bad ∧ B ∉ bad → ∃Rb. Nb = Rb$nb ∧ s.runs(Rb) = (Resp, A, B, [Na])) ∧

  -- actions:
  s'.runs := s.runs(Ra ↦ (Init, A, B, [Nb])) }

```

More precisely, if  $A$  or  $B$  is dishonest then  $Nb$  is arbitrary. Otherwise, there is a run  $Rb$  of responder  $B$  with initiator  $A$  that has generated the nonce  $Nb$  and previously received the nonce  $Na$ . This can be seen as  $A$  reading  $Nb$  from  $B$ 's store. We will eliminate this abstraction in the next refinement when we introduce communication channels. ♠

### 3.4.3 Refinements

We establish the secrecy and authentication properties of our guard protocol models by refining appropriately instantiated secrecy and authentication models from Section 3.3. In each case, we establish a refinement by reconstructing the abstract state (i.e., knowledge and authorization relations or signals) from the concrete one (i.e., the runs) and by identifying a pair of concrete events that refine the abstract ones (i.e., secret generation/learning and running/commit, respectively). The remaining events refine *skip*.

**Secrecy** We establish secrecy by refining the model  $s0$ . We therefore define relations  $knC(r)$  and  $azC(r)$ , which reconstruct the knowledge and authorization relations,  $kn$  and  $az$ , of  $s0$  from the runs  $r \in runsT$ . The simulation relation  $R_{s01}$  is  $\pi_{s01}^{-1}$ , where  $\pi_{s01}$  is the mediator function defined as follows.

$$\pi_{s01}(t) \equiv (\downarrow kn = knC(t.runs), az = azC(t.runs) \downarrow)$$

We can now explain how authorization guards are stated in terms of  $azC$ : we use the expression  $(d, A) \in azC(t.runs)$  to check whether an agent  $A$  is authorized to access data  $d$ . We will use authorization guards in Section 4.4 to model the confidential distribution of session keys.

**Authentication** We similarly refine  $a0i$  and  $a0n$  by reconstructing a signal multiset from the concrete runs. The simulation relation  $R_{a01}$  is  $\pi_{a01}^{-1}$ , where the mediator function  $\pi_{a01}$  is defined by

$$\pi_{a01}(t) \equiv (\!| sigs = sigsC(t.runs) |\!).$$

In general, for an agreement of agent  $A$  in role  $R$  with  $B$  in role  $S$  on data  $d$  with respect to agents  $h = [A, B, \dots]$ , the multiset  $sigsC(t.runs)$  contains a  $Commit(h, d)$  signal for each run of  $A$  in role  $R$  where the data  $d$  is known and a  $Running(h, d)$  signal for each run of  $B$  in role  $S$  knowing  $d$ . These are general patterns for establishing secrecy and authentication properties. Only the definitions of  $knC$ ,  $azC$ , and  $sigsC$  depend on the protocol.

In contrast to secrecy and authentication, we will formulate key freshness as a state predicate and establish it as an invariant of guard protocols (see Section 4.4.4).

**Example 3.6 (Refinement of authentication model).** Let  $a1$  be the model from Examples 3.4 and 3.5. The simulation relation  $R_{01}$  and mediator function  $\pi_{01}$  are as described for  $R_{a01}$  and  $\pi_{a01}$  above. It remains to define the function  $sigC$ , which maps completed initiator and responder runs to  $Commit$  and  $Running$  signals to express agreement on the nonces  $Na$  and  $Nb$ . Concretely, the function  $sigC(r) = m_r$  translates a run map  $r$  (such as  $runs$  in  $a1$ ) to the multiset of signals  $m_r$  (such as  $sigs$  in  $a0i$ ) that is defined as follows.

$$m_r(S) = \begin{cases} 1 & \text{if } S = Commit([A, B], (Ra\$na, Nb)) \text{ and } r(Ra) = (Init, A, B, [Nb]) \\ 1 & \text{if } S = Running([A, B], (Na, Rb\$nb)) \text{ and } r(Rb) = (Resp, A, B, [Na]) \\ 0 & \text{otherwise} \end{cases}$$

Note that, in general, the abstraction function  $sigC(r)$  maps each signal to the number of runs in a state that give rise to that signal (see Section 4). Whenever the run identifier appears in the signal's data (as in both cases above), there is at most one such run, since  $r$  is a partial function. Therefore, we directly use the constant 1 in the definition above instead of the more complex general definition.

At this point we can state and prove the refinement result, which establishes the entity authentication requirement **R2**. Its proof does not require auxiliary invariants, but relies on a number of basic properties of the function  $sigC$ .

**Proposition 3.7.**  $a1 \sqsubseteq_{R_{01}, \pi_{01}} a0i$ .

We have now satisfied both system requirements: injective agreement (**R2**) between an initiator and a responder (**R1**). By using abstraction we have captured the essential features of entity authentication protocols and established their main property once and for all. Recall that the channel protocols  $a2$  and  $c2$  are both refinements of  $a1$

channel type	dot notation	channel message	eavesdrop	fake
insecure	$A \rightarrow B : M$	$\text{Insec}(A, B, M)$	true	true
confidential	$A \rightarrow \bullet B : M$	$\text{Confid}(A, B, M)$	$A$ or $B$ bad	true
authentic	$A \bullet \rightarrow B : M$	$\text{Auth}(A, B, M)$	true	$A$ or $B$ bad
secure	$A \bullet \rightarrow \bullet B : M$	$\text{Secure}(A, B, M)$	$A$ or $B$ bad	$A$ or $B$ bad
confidential	$A \xrightarrow{K} \bullet B : M$	$\text{dConfid}(K, M)$	$K$ leaked	$K$ run-bounded
authentic	$A \bullet \xrightarrow{K} B : M$	$\text{dAuth}(K, M)$	true	$K$ leaked
secure	$A \bullet \xrightarrow{K} \bullet B : M$	$\text{dSecure}(K, M)$	$K$ leaked	$K$ leaked

Table 2: Channel notation, messages, and conditions for extraction and faking

(cf. Figure 2). Our proofs avoid the intricacies of an active attacker controlling communication. However, the resulting model is still quite abstract and requires further refinement to be executable in the intended hostile distributed environment. ♠

## 3.5 Level 2 — Channel protocols

At Level 2 of our refinement strategy, we introduce protocols that use communication channels with associated security properties. These channels carry plain text messages without cryptographic operations. We also introduce an explicit intruder acting in this distributed environment.

### 3.5.1 Channel messages

For informal use, we adopt the notation of [38] (second column of Table 2). We write  $A \rightarrow B$  for an insecure channel from agent  $A$  to agent  $B$ . The “:  $M$ ” indicates that the message  $M$  is sent on the channel. Security properties are indicated by a dot on one or both sides of the arrow. The respective agent has exclusive access to the marked end. A *confidential* channel  $A \rightarrow \bullet B$  provides a service to  $A$ :  $A$  knows that only  $B$  can receive messages. An *authentic* channel  $A \bullet \rightarrow B$  provides a service to  $B$ :  $B$  knows that only  $A$  can send messages. A *secure* channel  $A \bullet \rightarrow \bullet B$  provides both guarantees. Besides these static channels, we also use dynamic channels, which are accessed using a key  $K$ . For example,  $A \xrightarrow{K} \bullet B$  denotes a dynamic authentic channel. These keys are usually session keys generated during the protocol execution. Hence, the associated channels are dynamically created. Static and dynamic channels will later be realized by cryptographic operations using long-term and session keys, respectively.

We formalize the *channel messages* that can be transmitted by a data type *chmsg*, with constructors for static and dynamic channels. The first parameter of these constructors specifies the set of security properties as a combination of authenticity (auth) and confidentiality (confid). The actual payload message is a list of atomic messages.

$$\begin{aligned}
security &\triangleq \mathcal{P}(\{\text{auth, conf}\}) \\
chmsg &\triangleq \text{StatCh}(security, agent, agent, [atom]) \\
&\quad | \text{DynCh}(security, key, [atom])
\end{aligned}$$

Static channel messages name the sender and the receiver. For dynamic channel messages, names are replaced by a key, which determines access to the respective channel. Therefore, the agent names in the informal dot notation for dynamic channels (e.g., in  $A \xrightarrow{K} \bullet B$ ) only suggest the intended communication partners.

In our formal developments, we use the abbreviations given in the third column of Table 2. For example, we define

$$\text{Secure}(A, B, M) \equiv \text{StatCh}(\{\text{auth, conf}\}, A, B, M)$$

and call this a secure message from  $A$  to  $B$ . We also say that  $M$  is sent to  $B$  on a secure channel. We introduce analogous notions for the other channel messages.

### 3.5.2 Channel-based intruder

Based on the security attributes of channel messages we define the intruder capabilities for *eavesdropping* (or *extracting*) payload messages and for *faking* channel messages, indicated in the final two columns of Table 2. We formalize these intruder capabilities as two functions:

$$\begin{aligned}
extr_T &: \mathcal{P}(chmsg) \rightarrow \mathcal{P}(atom) \\
fake_{T,U} &: \mathcal{P}(chmsg) \rightarrow \mathcal{P}(chmsg)
\end{aligned}$$

where the parameter  $T \subseteq atom$  specifies the intruder's initial knowledge and the parameter  $U$  denotes a set of run identifiers. The expression  $extr_T(H)$  denotes the set of atoms that the intruder can extract from the set of (observed) messages  $H$  and  $fake_{T,U}(H)$  denotes the set of messages that the intruder can construct from messages in  $H$ . The faked messages carry payloads that are extracted from messages in  $H$  using  $extr_T(H)$ . The set of run identifiers  $U$  restricts the keys that the intruder can use to fake dynamic channel messages. These functions are defined by the rules in Figures 3 and 4. These rules state that the intruder can eavesdrop messages on non-confidential (i.e., insecure and authentic) channels and fake messages on non-authentic (i.e., insecure and confidential) channels. Moreover, the intruder can eavesdrop messages on confidential channels and fake messages on authentic channels, if these channels have a dishonest starting or ending point (static case) or the associated key  $K$  is known to the intruder (dynamic case).

The mild technical condition  $K \in rkey(U)$  in the third rule of Figure 4 restricts the intruder to using a session key in the set

$$rkey(U) \equiv \{R\$i \mid R \in U \wedge i \in \mathbb{N}\}$$

to fake a non-authentic dynamic message. We call these keys *run-bounded*. Below, we will use  $U = dom(s.runs)$  to preserve the invariant that all identifiers  $R\$i$  with  $R \notin dom(s.runs)$  are indeed fresh.

$$\begin{array}{c}
\frac{}{T \subseteq \text{extr}_T(H)} \\
\frac{\text{StatCh}(c, A, B, M) \in H \quad \text{confid} \notin c \vee A \in \text{bad} \vee B \in \text{bad}}{M \subseteq \text{extr}_T(H)} \\
\frac{\text{DynCh}(c, K, M) \in H \quad \text{confid} \notin c \vee K \in \text{extr}_T(H)}{M \subseteq \text{extr}_T(H)}
\end{array}$$

Figure 3: Rules defining extractable atoms

$$\begin{array}{c}
\frac{}{H \subseteq \text{fake}_{T,U}(H)} \\
\frac{M \subseteq \text{extr}_T(H) \quad \text{auth} \notin c \vee A \in \text{bad} \vee B \in \text{bad}}{\text{StatCh}(c, A, B, M) \in \text{fake}_{T,U}(H)} \\
\frac{M \subseteq \text{extr}_T(H) \quad (\text{auth} \notin c \wedge K \in \text{rkey}(U)) \vee K \in \text{extr}_T(H)}{\text{DynCh}(c, K, M) \in \text{fake}_{T,U}(H)}
\end{array}$$

Figure 4: Rules defining fakeable channel messages

The astute reader may be surprised that the intruder does not need to know a key to read from an authentic dynamic channel or write to a confidential dynamic channel. This situation differs from Dolev-Yao-style perfect cryptography. For example, verifying a MAC or signature requires a (shared or public) key. However, the key is typically only required for verifying the authenticity of the message and not for giving access to the authenticated message itself. Likewise, producing an encryption in a Dolev-Yao model requires a key. However, there are encryption schemes such as stream ciphers where the intruder can produce valid ciphertexts without knowing the encryption key.

Similarly, it may be surprising that we allow the intruder to (i) eavesdrop messages on confidential channels with a dishonest sender and (ii) send messages on authentic channels with a dishonest receiver. While in case (i), one can argue that the intruder knows the message anyway, we know no similar argument for case (ii). However, an important reason to allow this behavior is to enable implementations based on symmetric cryptography, for instance, realizing an authentic channel using MACs.

### 3.5.3 State, events, and refinement

Channel protocols extend the state of the guard protocol they refine with a variable *chan* containing a set of channel messages.

$$\Sigma_2 \triangleq \Sigma_1 + (\text{chan} \in \mathcal{P}(\text{chmsg}))$$

The protocol events use guards of the form  $M \in s.\text{chan}$  to receive a channel message  $M$ . These guards replace the non-local security guards in the guard protocols, which directly read other runs' local stores. Sending a message  $M$  is achieved by an action of the form  $s'.\text{chan} := s.\text{chan} \cup \{M\}$ .

Channel protocols include an intruder event, which closes the set of channel messages under fakeable messages.

$$fake_2 \equiv \{(s, s') \mid s'.chan := fake_{ik_0, dom(s.runs)}(s.chan)\}$$

Here, we work with the initial knowledge  $ik_0$  consisting of the sets of all agents, corrupted keys, and numbers.

$$ik_0 \equiv agent \mid corrKey \mid \mathbb{N}$$

The refinement of the abstract Level 1 model just extends the state record with the variable  $chan$ . Hence, the simulation relation is typically  $R_{12} \equiv \Pi_{runs}$ , stating that the values of variable  $runs$  of this and the previous model are identical.

**Example 3.8 (A channel-based authentication protocol).** The high level of abstraction of guard protocols allows many different realizations. Recall from the overview in Section 3.1 that we have also refined the model  $a1$  into two protocols, one using authentic and one using confidential channels, respectively. As an example, we now model the following abstract protocol using authentic channels.

$$\begin{aligned} \text{M1. } & A \rightarrow B : Na \\ \text{M2. } & B \bullet \rightarrow A : Nb, Na \end{aligned}$$

The initiator  $A$  sends the nonce  $Na$  to  $B$ , who returns it together with his own nonce  $Nb$  on an authentic channel. In Section 3.6, we will refine this protocol into the ISO/IEC 9798-3 two-pass unilateral authentication protocol [31].

The state  $\Sigma_{a2}$  of model  $a2$  is exactly as  $\Sigma_2$  described above. Initially, all fields are empty. The observation function is  $\pi_{runs}$ , i.e., it projects this state to the  $runs$  field.

The intruder event  $fake_{a2}$  is  $fake_2$  defined above. The protocol events send and receive messages to and from the insecure and authentic channels. In the first step,  $step1_{a1}$ , the initiator  $A$  sends  $Na$  on an insecure channel to  $B$ . In the second step, the responder  $B$  creates a new run identified by  $Rb$ , an associated nonce  $Nb$ , receives message M1 from the insecure channel, and authentically sends message M2 to  $A$ . This event adds the reception and sending of messages to the event  $step2_{a1}$  that it refines. In particular, instead of accepting any nonce  $Na$ , the nonce is now extracted from M1.

$$\begin{aligned} step2_{a2}(Rb, A, B, Na, Nb) &\equiv \{(s, s') \mid && \text{-- by } B, \text{ refines } step2_{a1} \\ &\text{-- guards:} && \\ &Rb \notin dom(s.runs) \wedge && \text{-- fresh run } Rb \\ &Nb = Rb\$nb \wedge && \text{-- fresh nonce } Nb \\ &Insec(A, B, [Na]) \in s.chan \wedge && \text{-- receive M1} \\ &\text{-- actions:} && \\ &s'.runs := s.runs(Rb \mapsto (Resp, A, B, [Na])) \wedge && \\ &s'.chan := s.chan \cup \{Auth(B, A, [Nb, Na])\} && \text{-- send M2} \end{aligned}$$

In the third step, the initiator run  $Ra$  receives message M2 and updates his local state with the nonce  $Nb$ .

```

step3a2(Ra,A,B,Na,Nb) ≡ {(s,s') |           -- by A, refines step3a1
  -- guards:
  s.runs(Ra) = (Init,A,B,[]) ∧
  Na = Ra$na ∧
  Auth(B,A,[Nb,Na]) ∈ s.chan ∧           -- recv M2

  -- actions:
  s'.runs := s.runs(Ra ↦ (Init,A,B,[Nb])) }

```

The reception of message M2 replaces the access to the responder run's memory of the refined event  $step3_{a1}$ . The corresponding guard refinement (**GRD**) of the refinement proof requires the following invariant, which states that authentic messages between honest agents indeed originate from an associated responder run identified by  $Rb$ .

```

autha2 ≡ {s | ∀A,B,Na,Nb.
  Auth(B,A,[Nb,Na]) ∈ s.chan ∧ B ∉ bad ∧ A ∉ bad
  → ∃Rb. Nb = Rb$nb ∧ s.runs(Rb) = (Resp,A,B,[Na]) }

```

This invariant is all that is needed to prove the refinement. Note that this is an internal invariant since it refers to channel messages, which are not observable.

Let  $a2$  be the above specification. The simulation relation is the canonical  $R_{12} ≡ \Pi_{runs}$  described above.

**Proposition 3.9.** *Let  $R'_{12}$  be the relation  $R_{12} \cap (\Sigma_{a1} \times auth_{a2})$ . Then  $reach(a2) \subseteq auth_{a2}$  and  $a2 \sqsubseteq_{R'_{12},id} a1$ .*

By using abstract channels, we retain the possibility of different cryptographic realizations. For example, we may realize the authentic channels using signatures or MACs. ♠

## 3.6 Level 3 — Cryptographic protocols

We model concrete protocols and the Dolev-Yao intruder using a standard theory of *cryptographic messages* due to Paulson [47]. At this level, messages are transmitted over insecure channels.

### 3.6.1 Cryptographic messages and setup

The type of messages,  $msg$ , is defined inductively from agents  $A \in agent$ , nonces  $N \in nonce$ , timestamps  $T \in \mathbb{N}$ , keys  $K \in key$ , pairs  $\{M_1, M_2\}$ , and encryptions  $\{M\}_K$ .

At this level, we can define a concrete type of long-term keys,  $ltk$ . As one of many possible examples, we may define  $ltk$  as follows.

$$ltk \triangleq pub(agent) \mid pri(agent) \mid shr(agent)$$

The terms  $pub(A)$ ,  $pri(A)$ , and  $shr(A)$  respectively denote  $A$ 's public key,  $A$ 's private key, and the symmetric key that  $A$  shares with the server  $S$ .

We can also concretize the previously declared abstract setup of cryptographic keys, *keySetup*. For example, we may define *keySetup* as follows.

$$\frac{A, B \in \text{agent}}{(\text{pub}(A), B) \in \text{keySetup}} \quad \frac{A \in \text{agent}}{(\text{pri}(A), A) \in \text{keySetup}} \quad \frac{A \in \text{agent} \quad C \in \{A, S\}}{(\text{shr}(A), C) \in \text{keySetup}}$$

As a consequence we can prove the following equation about *corrKey* (see Section 3.2).

$$\text{corrKey} = \text{ran}(\text{pub}) \cup \text{pri}(\text{bad}) \cup \text{shr}(\text{bad})$$

To formalize protocol properties and the intruder, we use the standard closure operators *parts*, *analz*, and *synth* on sets of messages (see [47]). The term *parts*(*H*) closes *H* under submessages (i.e., subterms of messages), *analz*(*H*) closes *H* under submessages accessible by projection and decryption using the keys in *H*, and *synth*(*H*) closes *H* under message compositions.

### 3.6.2 State and events

Cryptographic protocols replace the channel messages *chan* with a variable *IK* containing a set of cryptographic messages. Therefore, like the channel protocols, they extend the state  $\Sigma_1$  of the guard protocol they refine.

$$\Sigma_3 \triangleq \Sigma_1 + (\{ IK \in \mathcal{P}(\text{msg}) \})$$

Initially, the set *IK* contains the initial intruder knowledge, e.g., the long-term keys of all bad agents, *corrKey*. Note that  $\text{agent} \cup \mathbb{N} \subseteq \text{synth}(H)$  for all sets of messages *H*.

Protocol events receive messages by using guards of the form  $M \in s.IK$  and send messages by actions of the form  $s'.IK := s.IK \cup \{M\}$ . The Dolev-Yao intruder can generate and send messages from the set  $\text{synth}(\text{analz}(s.IK))$ .

$$\text{fake}_3 \equiv \{(s, s') \mid s'.IK := \text{synth}(\text{analz}(s.IK))\}$$

**Example 3.10 (A cryptographic authentication protocol).** We now refine the abstract protocol model *a2* into the ISO/IEC 9798-3 two-pass unilateral authentication protocol [31] by translating the authentic channels into signed messages communicated over an insecure channel.

$$\begin{aligned} \text{M1. } & A \rightarrow B : A, B, Na \\ \text{M2. } & B \rightarrow A : \{\{ Nb, Na, A \}\}_{\text{pri}(B)} \end{aligned}$$

The state  $\Sigma_{\text{iso3}}$  is  $\Sigma_3$  described above. Initially, the *runs* field is empty and *IK* = *corrKey*. Note that for this example the shared keys could be omitted. This models Assumptions **A2** and **A3** about static corruption and the cryptographic setup. Only the *runs* field is observable.

We restrict our presentation to the second protocol step, *step2<sub>iso3</sub>*. It refines the abstract event *step2<sub>a2</sub>* by replacing the insecure message M1 by a corresponding message in *IK* and the authentic message M2 by a signed message in *IK*.

```

step2iso3(Rb,A,B,Na,Nb) ≡ {(s,s') |           -- by B, refines step2a2
  -- guards:
  Rb ∉ dom(s.runs) ∧
  Nb = Rb$nb ∧
  ⌊A,B,Na⌋ ∈ s.IK ∧                               -- receive M1

  -- actions:
  s'.runs := s.runs(Rb ↦ (Resp,A,B,[Na])),
  s'.IK := s.IK ∪ {⌊Nb,Na,A⌋pri(B)} }          -- send M2

```

The intruder event  $fake_{iso3}$  is  $fake_3$  introduced above, which models the Dolev-Yao intruder described in Assumption **A1**. ♠

### 3.6.3 Refinement

The refinement of channel protocols by cryptographic ones is based on a protocol-dependent message abstraction function  $absMsg : \mathcal{P}(msg) \rightarrow \mathcal{P}(chmsg)$ . Given such a function, the simulation relation  $R_{23}$  is defined as the intersection of the following four relations.

$$\begin{aligned}
R_{23}^{msgs} &\equiv \{(s,t) \mid absMsg(parts(t.IK)) \subseteq s.chan\} \\
R_{23}^{non} &\equiv \{(s,t) \mid analz(t.IK) \cap nonce \subseteq extr_{ik_0}(s.chan)\} \\
R_{23}^{key} &\equiv \{(s,t) \mid analz(t.IK) \cap key \subseteq extr_{ik_0}(s.chan)\} \\
R_{23}^{pres} &\equiv \Pi_{runs}
\end{aligned}$$

The relation  $R_{23}^{msgs}$  expresses that the abstractions of concrete message parts in  $t.IK$  are contained in the channel variable  $s.chan$ . Abstracting message parts instead of the messages themselves increases the flexibility of the abstractions. We will further discuss this point in Section 4.6 where this possibility enables the modification of the communication topology between Levels 2 and 3. The relations  $R_{23}^{key}$  and  $R_{23}^{non}$  state that the abstract intruder knows at least the nonces and keys that the concrete intruder also knows. If no keys or no nonces appear in a protocol then the corresponding relation can be dropped from the simulation relation. Finally, the relation  $R_{23}^{pres}$  states that the variable  $runs$  is preserved, i.e., it has the same value in the abstract and concrete model. In concrete applications, this relation may include other preserved state variables.

In the refinement proof for the intruder events, a variant of the following action refinement (**ACT**) proof obligation arises.

$$(s(\llbracket chan := fake(s.chan) \rrbracket), t(\llbracket IK := synth(analz(t.IK)) \rrbracket)) \in R_{23}$$

This states that the successor states resulting from the respective intruder actions are still in the simulation relation. The proof of the part concerning  $R_{23}^{msgs}$  relies on the property

$$(s,t) \in R_{23} \Rightarrow absMsg(parts(synth(analz(t.IK)))) \subseteq fake(s.chan),$$

which typically follows from the definitions of  $absMsg$  and  $fake$  and from general properties of  $parts$ ,  $synth$ , and  $analz$ .

**Example 3.11.** The simulation relation in this refinement is  $R_{23}$  as defined above, except that  $R_{23}^{key}$  is not needed. We concretize the message abstraction function  $absMsg(H)$  as follows.

$$\frac{\{A, B, Na\} \in H}{\text{Insec}(A, B, [Na]) \in absMsg(H)} \quad \frac{\{Nb, Na, A\}_{pri(B)} \in H}{\text{Auth}(B, A, [Nb, Na]) \in absMsg(H)}$$

The refinement proof only requires one internal invariant expressing the secrecy of the private signing keys.

$$keys_{iso3} \equiv \{s \mid \forall A. pri(A) \in analz(s.IK) \rightarrow A \in bad\}$$

We can now state the refinement result relating this model of the ISO/IEC 9798-3 protocol to the abstract channel model  $a2$ .

**Proposition 3.12.** *Let  $R'_{23}$  be the relation  $R_{23} \cap \Sigma_{a2} \times keys_{iso3}$ . Then  $reach(iso3) \sqsubseteq keys_{iso3}$  and  $iso3 \sqsubseteq_{R'_{23}, id} a2$ .  $\spadesuit$*

### 3.7 Discussion

We have presented our four-level refinement strategy along with its supporting infrastructure. We have illustrated our approach with the development of simple entity authentication protocols. This resulted in two different concrete protocols at Level 3: ISO/IEC 9798-3 and NSL/2. We satisfied all system requirements by proving properties of the abstract model at Levels 0 and 1. As these models are not directly implementable, we continued our refinements, thereby obtaining models that are suitable for an implementation in the intended hostile distributed environment and, crucially, inherit the properties we proved for the abstract models. The simulation relation and invariants used here at Levels 2 and 3 are canonical for our refinement strategy (cf. Section 4). We would like to emphasize that a number of alternative cryptographic realizations of the channel protocols are possible, for example, using symmetric encryption or MACs. Our approach fosters abstraction and enables the sharing of structure and proofs.

## 4 Key Establishment Protocols

In this section, we validate our refinement approach by developing a family of key establishment protocols, including the Needham-Schroeder Shared Key (NSSK), core versions of the Kerberos 4 and 5 protocols, and the Denning-Sacco protocol. Compared to the simple running examples from Section 3, our protocol models feature additional elements such as timestamps, replay caches, dynamic channels, and a changing communication structure. We also prove additional security properties related to session keys, such as key confirmation, key freshness, and key recentness.

## 4.1 Requirements and Assumptions

Our informal requirements and assumptions for (server-based) key establishment protocols follow below. The first three requirements are mandatory and must be satisfied by all protocols we consider. The last three requirements are optional. We will formalize these requirements in subsequent sections.

**Requirement R1 (Key distribution).** The server generates and distributes a fresh session key to an initiator and a responder.

**Requirement R2 (Key secrecy).** Only authorized agents may learn a session key, unless one of them is dishonest whereby other agents may also learn it.

The next two requirements cover authentication properties, which we will formalize in Section 4.3 as injective or non-injective agreements.

**Requirement R3 (Server authentication).** The initiator and the responder each authenticate the server on the session key and possibly on additional data.

**Requirement R4 (Key confirmation).** The initiator and the responder authenticate each other on the session key and possibly on additional data, thereby confirming to each other their knowledge of the key.

Two additional (and independent) requirements concern the freshness and recentness of the session key. A key is *fresh* if it is only used in a single session and is *recent* if its lifetime does not exceed a specified limit.

**Requirement R5 (Key freshness).** The initiator and responder obtain assurance that the session key is fresh.

**Requirement R6 (Key recentness).** The initiator and responder obtain assurance that the session key is recent.

The environment assumptions about the intruder, static corruption, and the cryptographic setup are the same as in Section 3.1.

## 4.2 Development Overview

We concretize our refinement strategy for deriving different server-based key establishment protocols: the Needham-Schroeder Shared-Key (NSSK) protocol [44], the Denning-Sacco protocol [29], and core versions of Kerberos 4 [51] and Kerberos 5 [45]. The refinement graph in Figure 5 summarizes our development. Recall that each node in such a graph represents a model and each arc  $m \rightarrow m'$  represents a refinement  $m \sqsubseteq_{\pi} m'$  for a given mediator function  $\pi$ , not shown. The superscripts refer to the requirements established, where  $i$  and  $r$  denote the initiator and responder.

At Level 0, we have the abstract models of secrecy ( $s0$ ) and authentication ( $a0i$ ,  $a0n$ ) from Section 3.3. At Level 1, our first guard protocol,  $kt1$ , abstractly models server-based secret key transport (**R1**, **R2**). It refines the secrecy model  $s0$  and is an ancestor of all key transport protocols that we have derived. The model  $kt1$  therefore

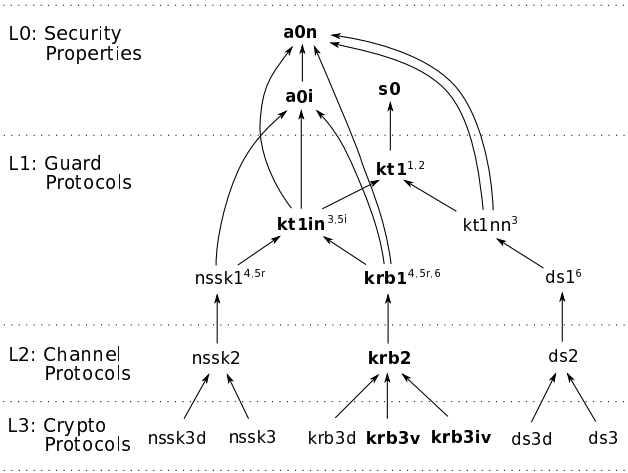


Figure 5: Refinement graph

protocol	model	R1	R2	R3	R4	R5	R6
NSSK	$nssk3$	✓	✓	i/n	i/i	✓	-
Kerberos 4	$krb3iv$	✓	✓	i/n	n/i	✓	✓
Kerberos 5	$krb3v$	✓	✓	i/n	n/i	✓	✓
Denning-S.	$ds3$	✓	✓	n/n	-	-	✓

Table 3: L3 models and their properties

provides secret key distribution, but does not guarantee the session key’s authenticity, freshness, or recentness. Hence, we refine  $ktI$  into further guard protocols that establish authentication properties (**R3**, **R4**) and use *freshness identifiers* [30], namely, nonces or timestamps, to prevent replays and guarantee key freshness and recentness (**R5**, **R6**). We do this in two stages.

In the first stage, we refine  $ktI$  into two different models that realize server authentication (**R3**). In the first model,  $ktIin$ , the initiator injectively agrees with the server on the session key, while the responder non-injectively agrees with the server. The injective agreement and secrecy entail key freshness (**R5**) for the initiator. In the second model,  $ktInn$ , both initiator and responder achieve non-injective agreements with the server. The data agreed upon, in addition to the session key, is a parameter in these models. We establish each authentication property by refining an L0 model,  $a0n$  or  $a0i$ , using a different mediator function. This explains why there are multiple paths between some models in Figure 5.

In the second stage, we refine the model  $ktIin$  into  $nssk1$  and  $krb1$  and establish key confirmation (**R4**). We achieve this by adding protocol steps and proving mutual agreement between the initiator and the responder on the key and other data. In  $nssk1$ ,

these agreements are injective due to the use of nonces. In *krb1*, we use timestamps to ensure key recentness (**R6**). A replay prevention cache allows the responder to obtain an injective agreement with the initiator. Key freshness (**R5**) for the responder relies on both authentication and secrecy properties. Finally, we also refine the model *kt1nn* into *ds1* using timestamps to obtain key recentness. At this point, all requirements are established. The remaining two levels realize the environment assumptions **A1-A3**, making the protocol fit for execution in a hostile distributed environment.

At Level 2, we construct the three channel-based models *nssk2*, *krb2*, and *ds2*, where the roles exchange channel messages instead of reading each other’s memory. The server distributes the session key on static secure channels to the initiator and responder and key confirmation is realized in *nssk2* and *krb2* using dynamic authentic channels protected by the session key.

At Level 3, we replace the channel messages by cryptographic messages sent over an insecure channel. We implement the static secure channels by symmetric encryption with long-term keys and the dynamic authentic channels by encryption with the session key. The models at this level differ in their handling of the ciphertext containing the responder’s session key (called a *ticket*). In models with names ending in *d*, namely, *nssk3d*, *ds3d*, and *krb3d*, the server sends the ticket directly to the responder. In the other models, the communication topology changes: the server sends the ticket to the initiator who forwards it to the responder. While in *krb3v* the ticket is sent alongside the ciphertext containing the initiator’s session key, it appears inside the ciphertext in the models *nssk3*, *ds3*, and *krb3iv*.

Table 3 summarizes the requirements achieved by the final protocols. In the columns for the authentication requirements **R3** and **R4**, ‘i’ means injective and ‘n’ means non-injective agreement. The slash separates initiator and responder guarantees. The models with names ending in *d* (not listed in the table) achieve the same properties as their listed siblings.

Based on the modeling and reasoning framework and the infrastructure from Section 3, in the following sections we develop concrete models at each abstraction level. We focus on the models typeset in boldface in Figure 5 leading to the core versions of Kerberos. Figure 6 provides an overview of most refinements between these models and the related propositions. State spaces (without variable types) and simulation relations are displayed on the left hand side, observations and mediator functions on the right hand side, and observation functions are shown as left-to-right arrows. Triples of the form  $(i, r, s)$  describe the frames of completed initiator, responder, and server runs, where ... stands for the fields inherited from the model above. A star (\*) means that the related refinement proof requires invariants to strengthen the simulation relation. In this figure, we have not included the refinements of the authentication models *a0n* and *a0i* by the models *kt1in* and *krb1* (Propositions 4.2 and 4.5). These exhibit a structure similar to the refinement of *s0* by *kt1* (cf. Section 3.4). We have also omitted the refinement of *krb2* into core Kerberos 4 (*krb3iv*) stated in Proposition 4.9 as it is similar to the refinement into core Kerberos 5 (*krb3v*). This figure is primarily intended as a reference for the reader, but we will return to it in Section 4.7.1, where we discuss the security guarantees that the refinements yield for the final models at Level 3.

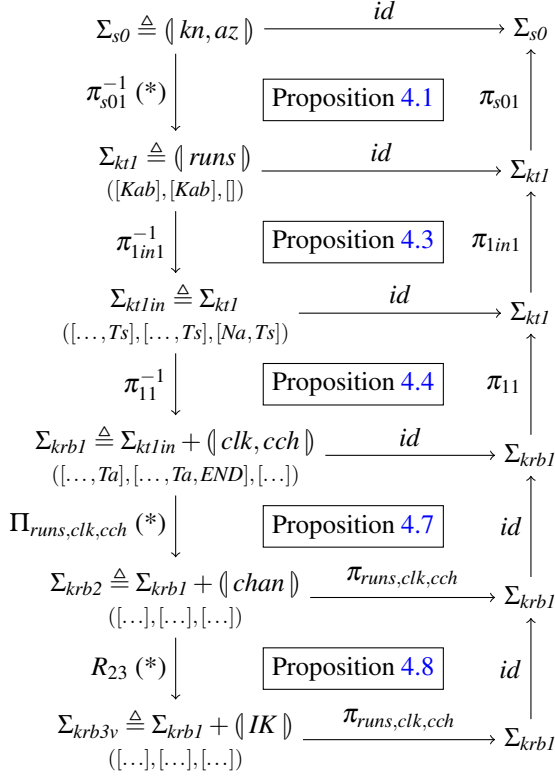


Figure 6: Details of refinements in the Kerberos development.

### 4.3 Security Properties (L0)

We start our development by formalizing the security requirements. We formalize each secrecy and authentication requirement as an instance of the corresponding Level-0 model from Section 3.3. We will later show that our guard protocol models (L1) refine these instantiated models, thus establishing the respective requirements (by Proposition 2.4). We will formalize key freshness and key recentness as invariants of Level 1 protocols and therefore discuss these later in Section 4.4.

**Secrecy** The instantiation of the polymorphic type of data of the model  $s0$  to keys provides an abstract model of key distribution and key secrecy. Refining this model will establish the Requirements **R1** and **R2**.

**Authentication** We formalize the Requirements **R3** and **R4**. For this purpose, we must specify the data agreed upon. We use *authentication graphs* to represent this information visually. Figure 7 displays the authentication graph of the Kerberos protocols. In these graphs, there is one node for each protocol role. Each node is labeled

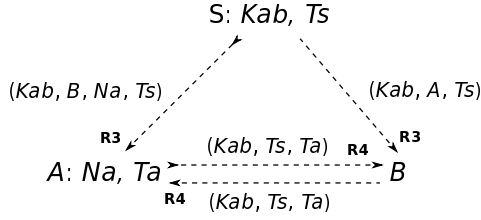


Figure 7: Authentication graph for Kerberos

by an agent name (in the given role), possibly followed by a list of freshness identifiers generated during the role’s execution. For example, the server  $S$  generates the session key  $Kab$  and a timestamp  $Ts$ , and the initiator generates a nonce  $Na$  and a timestamp  $Ta$ . Each arrow specifies one agreement property by defining the parameters  $h$  and  $d$  for the *running* and *commit* events of models  $a0n$  or  $a0i$ . The arrow endpoints define the agents  $h$ , whose honesty is assumed, and the tuples labeling the arrows specify the data  $d$  to be agreed upon between these agents. Note that for the current development, we do not need to assume the honesty of agents other than the participants in the agreement. An arrow tail indicates an injective agreement. The boldface labels indicate the requirements that are established for the agent near the arrow head. For example, the arrow from  $S$  to  $B$  labeled by  $(Kab, A, Ts)$  means that the responder  $B$  non-injectively agrees with the server on  $(Kab, A, Ts)$ , assuming the honesty of  $S$  and  $B$  (**R3**). The arrow from  $A$  to  $B$  labeled by  $(Kab, Ts, Ta)$  means that  $B$  injectively agrees with  $A$  on  $Kab, Ts$ , and  $Ta$ , assuming the honesty of  $A$  and  $B$  (**R4**).

To prove that an L1 model establishes an agreement of role  $R$  with role  $S$ , we identify an event of role  $R$  that refines the *commit* event and an event of role  $S$  that refines the *running* event. All other events must refine *skip*. Each agreement requires a different mapping of the protocol events to the *running* and *commit* events and therefore requires a separate refinement of the model  $a0i$  or  $a0n$  (cf. Figure 5).

## 4.4 Guard Protocols (L1)

In our server-based key transport protocols, there are three roles: a key-generating *server* and key-receiving *initiators* and *responders*. The state records runtime information about the execution of these roles as described in Section 3.4.

$$\Sigma_{kt1} \triangleq (\text{runs} \in \text{runs}T)$$

Initially, the *runs* map is empty. The entire state is observable, i.e., the observation function is the identity.

### 4.4.1 Secret key distribution

A sequence chart of our first abstract key transport protocol model, *kt1*, appears in Figure 8. This model establishes **R1** and **R2** as follows. The server  $S$  generates the session key  $Kab$ , which is indicated by the role label  $S: Kab$ . The initiator  $A$  and the

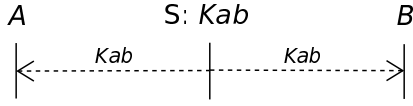


Figure 8: Basic secret key distribution (*ktI*)

responder  $B$  then *secretly* acquire this key and record it in their run frames, which is represented by arrows from  $S$  to  $A$  and  $B$  labeled by  $K_{ab}$ . These arrows do *not* represent the communication of messages, since there are no messages or channels at this stage.

Before presenting the events of the specification *ktI*, we discuss the simulation relation used in the refinement of the model *s0* from Section 3.3.1, which establishes session key secrecy. As described in Section 3.4, we define relations  $knC(r)$  and  $azC(r)$ , which reconstruct *s0*'s knowledge and authorization relations from the runs  $r \in runsT$ .

We define the relation  $knC(r)$  by four rules. We give two examples describing the initiator and server's session key knowledge. There is a similar rule for the responder.

$$\frac{r(Ra) = (\text{Init}, A, B, K\#ns)}{(K, A) \in knC(r)} \quad \frac{r(Rs) = (\text{Serv}, A, B, ns)}{(Rs\$sk, S) \in knC(r)}$$

Here,  $Rs\$sk$  is the fresh value used by the server run  $Rs$  for the session key and  $sk$  is an arbitrary natural number. An additional rule states that the initial key setup is contained in the knowledge relation, i.e.,  $keySetup \subseteq knC(r)$ .

The following rule defines who is authorized to learn a session key that the server  $S$  generated for  $A$  and  $B$ , namely  $A$ ,  $B$ , and  $S$ , if  $A$  and  $B$  are honest, and everyone otherwise.

$$\frac{r(Rs) = (\text{Serv}, A, B, ns) \quad C \in \{A, B, S\} \vee A \in bad \vee B \in bad}{(Rs\$sk, C) \in azC(r)} \quad (1)$$

Two additional rules state that  $keySetup \subseteq azC(r)$  and that anyone is authorized to learn corrupted keys.

The specification *ktI* has five events, each modeling a protocol step. The first event creates a new run  $Ra$  of initiator  $A$  with responder  $B$  by updating  $runs$  with  $(Ra \mapsto (\text{Init}, A, B, []))$ . The second event creates a responder run analogously. These two events refine *skip*. In the third event, we generate a new server run  $Rs$  with associated fresh session key  $Kab$ . This event refines  $gen_{s0}(Kab, S, [S, A, B])$ .

$$\begin{aligned} step3_{ktI}(Rs, A, B, Kab) \equiv \{ & (s, s') \mid \quad \text{-- by } S, \text{ refines } gen_{s0} \\ & Rs \notin dom(s.runs) \wedge \quad \text{-- fresh server run} \\ & Kab = Rs\$sk \wedge \quad \text{-- session key} \\ & s'.runs := s.runs(Rs \mapsto (\text{Serv}, A, B, [])) \} \end{aligned}$$

The final two events model the confidential acquisition of the session key by the initiator and the responder. They both refine the event  $learn_{s0}$ . In Step 5, the responder  $B$  acquires the session key  $Kab$  in its run  $Rb$ .

$$\begin{aligned} step5_{ktI}(Rb, A, B, Kab) \equiv \{ & (s, s') \mid \quad \text{-- by } B, \text{ refines } learn_{s0} \\ & s.runs(Rb) = (\text{Resp}, A, B, []) \wedge \quad \text{-- } B's \text{ run} \} \end{aligned}$$

$$(Kab, B) \in azC(s.runs) \wedge \quad \text{-- check authorization}$$

$$s'.runs := s.runs(Rb \mapsto (Resp, A, B, [Kab])) \}$$

The first guard requires that  $Rb$  identifies a run of responder  $B$  with initiator  $A$ , where  $B$  has not yet received a key. The action updates the responder run with the session key  $Kab$ . The initiator's  $step4_{kt1}$  is analogous.

The second guard is an authorization guard requiring that  $B$  is authorized to learn  $Kab$ . There are two cases according to the definition of  $azC$ . The first case, described by rule (1), corresponds to reading a (session) key  $Kab$  from the server, who determined the authorization to access the key. Note that there is no guarantee that the key was generated for  $B$ . In the second case,  $Kab$  is a static key, which may be corrupted. For now, there are no further constraints on  $Kab$ . The authorization guard is sufficient to preserve the secrecy of  $Kab$ . In Section 4.4.2, we will establish authentication properties to ensure that honest agents only accept session keys generated for them and shared with the intended partner.

Instantiating the simulation relation  $R_{s01}$  from Section 3.4 with the relations  $azC(r)$  and  $knC(r)$  defined above, we show that the model  $kt1$  refines the secrecy model  $s0$ . The guard strengthening proof in the refinement of the event  $gen_{s0}$  by  $step3_{kt1}$  requires an invariant,  $key_{kt1}$ , stating that no fresh key  $K$  is in the domain of either  $knC(s.runs)$  or  $azC(s.runs)$ .

**Proposition 4.1.** Let  $R'_{s01} \equiv R_{s01} \cap (\Sigma_{s0} \times key_{kt1})$ . Then we have  $reach(kt1) \subseteq key_{kt1}$  and  $kt1 \sqsubseteq_{R'_{s01}, \pi_{s01}} s0$ .

Since the abstract variables  $kn$  and  $az$  are observable and reconstructable from the concrete state, the secrecy invariant for  $s0$  (Proposition 3.1) is inherited by  $kt1$  (Corollary 2.8), which thus realizes secret key distribution (**R1**, **R2**).

#### 4.4.2 Server authentication

We now refine the model  $kt1$  into  $kt1in$  and establish agreements of the initiator and the responder with the server on the session key and additional data. The additional data is a parameter of  $kt1in$ . However, for the sake of the presentation, we will focus our attention on the instantiation of  $kt1in$  for the Kerberos development. The models  $kt1$  and  $kt1in$  have identical state spaces, but in  $kt1in$  we introduce nonces and timestamps, which are part of the data included in the agreement and recorded in the run frames of the different roles. The observation function is the identity. Figure 9 shows a sequence chart for this model. The labels below the arrows denote agreements and an arrow tail indicates an injective agreement as specified in Figure 7.  $Na$  is a nonce generated by  $A$  and  $Ts$  is a freshness identifier generated by  $S$  that we will later refine into a timestamp (i.e., a clock reading). The initiator  $A$  achieves (**R3**) by an injective agreement with the server on  $(Kab, B, Na, Ts)$  and the responder  $B$  establishes (**R3**) by a non-injective agreement with the server on  $(Kab, A, Ts)$ . The model  $kt1in$  refines  $kt1$ ,  $a0i$ , and  $a0n$  (cf. Figures 5 and 6). We establish key freshness (**R5**) for the initiator as an invariant (cf. Section 4.4.4 for the similar responder case).

We obtain the model  $kt1in$  by modifying  $kt1$  in two ways. First, we introduce new event parameters and corresponding state updates to reflect that both partners of the

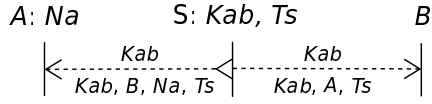


Figure 9: Adding server authentication (*ktlin*)

agreement know the data being agreed upon. In the server's Step 3, we add a nonce  $Na$  and the freshness identifier  $Ts$  to the parameters and record these in the run frame, i.e., the runs are updated with  $Rs \mapsto (\text{Serv}, A, B, [Na, Ts])$ . Neither  $Na$  nor  $Ts$  are constrained by any guards. In the responder's Step 5, we add the parameter  $Ts$  and update the runs with  $Rb \mapsto (\text{Resp}, A, B, [Kab, Ts])$  and similarly in the initiator's Step 4.

Second, we realize the agreements described above by adding authentication guards to the key-receiving Steps 4 and 5. These guards may either be added directly to the respective events or *discovered* during the refinement proof of the *commit* event of the model  $a0n$  or  $a0i$ . Here, we describe guard discovery.

For the refinement of  $a0n$ , we therefore first define the function  $\text{sig}^{Crs}$ , which reconstructs signal multisets from protocol runs, and use it to obtain the mediator function  $\pi_{a01}^{rs}$  and simulation relation  $R_{a01}^{rs}$ , as described in Section 3.4. The multi-set  $\text{sigs}^{Crs}(r) \equiv m_r$  contains a Commit signal for each completed responder run. We formalize this as follows.

$$m_r(\text{Commit}([B, S], (Kab, A, Ts))) \equiv |RR| \\ \text{where } RR \equiv \{Rb \mid \exists nl. r(Rb) = (\text{Resp}, A, B, Kab\#Ts\#nl)\}$$

Similarly, completed server runs give rise to Running signals. Since the session key  $Kab$  is derived from the (unique) server run identifier  $Rs$ , a simpler definition suffices.

$$m_r(\text{Running}([B, S], (Kab, A, Ts))) \equiv \\ \text{if } \exists Rs, Na, nl. (Kab = Rs\$sk \wedge r(Rs) = (\text{Serv}, A, B, Na\#Ts\#nl)) \text{ then } 1 \text{ else } 0$$

The existential quantifications on  $nl$  account for extensions to the run frames with additional atomic messages in later refinements. Finally, we set  $m_r(x) \equiv 0$  at all other points  $x$ .

Next, we prove that the server's event  $\text{step}3_{ktlin}$  and the responder's event  $\text{step}5_{ktlin}$  refine the events  $\text{running}_{a0n}([B, S], (Kab, A, Ts))$  and  $\text{commit}_{a0n}([B, S], (Kab, A, Ts))$  of the abstract model  $a0n$ . The remaining events refine *skip*. In the proof of guard refinement (**GRD**) for  $\text{step}5_{ktlin}$ , we get stuck in a proof state that directly suggests the following authentication guard for this event.

$$B \notin \text{bad} \rightarrow \exists Rs, Na, nl. (Kab = Rs\$sk \wedge s.\text{runs}(Rs) = (\text{Serv}, A, B, Na\#Ts\#nl)) \quad (2)$$

This guard guarantees to an honest  $B$  that there is a server in a state counting as a matching  $\text{Running}([B, S], (Kab, A, Ts))$  signal. After adding this guard, the proof succeeds.

For the refinement of  $a0i$ , we similarly discover the authentication guard for the event  $\text{step}4_{ktlin}(Ra, A, B, Na, Kab, Ts)$  in the proof that this event refines the abstract event  $\text{commit}_{a0i}([A, S], (Kab, B, Na, Ts))$ .

$$A \notin \text{bad} \rightarrow \exists Rs, nl. (Kab = Rs\$sk \wedge s.\text{runs}(Rs) = (\text{Serv}, A, B, Na\#Ts\#nl))$$

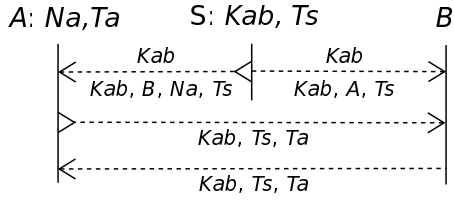


Figure 10: Adding key confirmation (*krb1*)

Compared to (2), the absence of the existential quantification on  $Na$  reflects that this agreement includes  $Na$ .

**Proposition 4.2.**  $ktlin \sqsubseteq_{\pi_{a01}^{is}} a0i$  for the initiator and  $ktlin \sqsubseteq_{\pi_{a01}^{rs}} a0n$  for the responder.

Finally, it is easy to see that  $ktlin$  refines  $ktl$ . The mediator function  $\pi_{1in1}$  removes the nonce  $Na$  and the timestamp  $Ts$  from server run frames and  $Ts$  from initiator and responder run frames, therefore only keeping the session key  $Kab$ .

**Proposition 4.3.**  $ktlin \sqsubseteq_{\pi_{1in1}^{-1}, \pi_{1in1}} ktl$ .

#### 4.4.3 Key confirmation

We next extend the model  $ktlin$  to an abstract model of Kerberos (Figure 10), which achieves key confirmation (R4), key freshness for the responder (R5), and key recentness (R6). To model timestamps and their expiration, we explicitly introduce a (discrete-time) global clock. For key recentness, the initiator and responder check the validity of a timestamp  $Ts$  that the server associates with the session key  $Kab$ . For key confirmation, the initiator and responder mutually agree on the session key  $Kab$ , its associated timestamp  $Ts$ , and an initiator timestamp  $Ta$  (cf. Figure 7). The responder caches keys  $Kab$  and timestamps  $Ta$  to obtain an injective agreement with the initiator. We assume arbitrary fixed lifetimes  $Ls$  and  $La$  for server and initiator timestamps. Note that the sequence chart in Figure 10 contains all agreements specified in Figure 7 and (partially) orders them causally.

We extend the state of  $ktl$  with two additional variables, reflecting the elements discussed above.

$$\Sigma_{krb1} \triangleq \Sigma_{ktl} + (\text{clk} \in \text{time}, \text{cch} \in \mathcal{P}(\text{agent} \times \text{key} \times \text{time}))$$

The variable  $clk$  models the discrete-time clock. We introduce an associated  $tick(T)$  event that increments the clock by  $T$  time units. All other events are assumed to take no time and hence do not modify the clock. The variable  $cch$  represents a cache storing triples  $(B, Kab, Ta)$  consisting of an agent name  $B$ , a session key  $Kab$ , and an initiator timestamp  $Ta$ . For replay protection, a responder  $B$  checks the cache before accepting a key  $Kab$  with timestamp  $Ta$ . A new  $purge(B)$  event removes from  $B$ 's cache those entries whose timestamps  $Ta$  have expired and thus are no longer valid, which is the case when  $s.clk \geq Ta + La$ .

The events for Steps 1 to 5 of the model *krb1* are derived from the corresponding *kt1in* events, possibly adding guards and actions. In Step 3, we turn *Ts* into a timestamp by adding the guard  $Ts = s.clk$ .

In the initiator's Step 4, we add the initiator's timestamp *Ta* as a parameter and record it in the frame, and we introduce two time-related guards.

$$\begin{aligned}
step4_{krb1}(Ra, A, B, Na, Kab, Ts, Ta) &\equiv \{(s, s') \mid \text{-- by } A \\
&\dots \text{-- guards of } step4_{kt1in} \text{ (omitted)} \\
Ta = s.clk \wedge &\quad \text{-- get timestamp} \\
s.clk < Ts + Ls \wedge &\quad \text{-- chk validity of } Ts \\
s'.runs := s.runs(Ra \mapsto &(\text{Init}, A, B, [Kab, Ts, Ta])) \}
\end{aligned}$$

The first guard states that the timestamp *Ta* is the current value of the clock. The second guard ensures the validity of the server timestamp *Ts*.

In the responder's Step 5, we also add *Ta* as a parameter and record it in the frame. Furthermore, we introduce four new guards and a new action.

$$\begin{aligned}
step5_{krb1}(Rb, A, B, Kab, Ts, Ta) &\equiv \{(s, s') \mid \text{-- by } B \\
&\dots \text{-- guards of } step5_{kt1in} \text{ (omitted)} \\
&\text{-- for agreement with } A \text{ on } (Kab, Ts, Ta) \\
(A \notin bad \wedge B \notin bad \rightarrow \exists Ra, nl. s.runs(Ra) = &(\text{Init}, A, B, Kab\#Ts\#Ta\#nl)) \wedge \\
(B, Kab, Ta) \notin s.cch \wedge &\quad \text{-- replay protection} \\
s.clk < Ta + La \wedge &\quad \text{-- chk validity of } Ta \\
s.clk < Ts + Ls \wedge &\quad \text{-- chk validity of } Ts \\
s'.cch := s.cch \cup \{(B, Kab, Ta)\} &\quad \text{-- cache update} \\
s'.runs := s.runs(Rb \mapsto &(\text{Resp}, A, B, [Kab, Ts, Ta])) \}
\end{aligned}$$

The first guard ensures agreement with the initiator on the data  $(Kab, Ts, Ta)$ . The second guard achieves injectivity for the responder by checking that *B* has not previously seen *Kab* with timestamp *Ta*. The last two guards ensure recentness by checking the validity of *Ta* and *Ts*. The new action adds  $(B, Kab, Ta)$  to the cache to avoid future replays.

Finally, we add a new Step 6 to the initiator, which uses an authentication guard to achieve agreement with a responder run *Rb* on *Kab*, *Ts*, and *Ta*. We add an arbitrary value *END* to the frame to mark the initiator run's termination.

$$\begin{aligned}
step6_{krb1}(Ra, A, B, Na, Kab, Ts, Ta) &\equiv \{(s, s') \mid \text{-- by } A \\
s.runs(Ra) = (\text{Init}, A, B, [Kab, Ts, Ta]) \wedge & \\
&\text{-- for agreement with } B \text{ on } (Kab, Ts, Ta) \\
A \notin bad \wedge B \notin bad \rightarrow (\exists Rb. s.runs(Rb) = &(\text{Resp}, A, B, [Kab, Ts, Ta])) \\
s'.runs := s.runs(Ra \mapsto &(\text{Init}, A, B, [Kab, Ts, Ta, END])) \}
\end{aligned}$$

The mediator function  $\pi_{11}$  in the refinement of *kt1in* by *krb1* drops the timestamps *Ta* and the termination marker *END* from initiator and responder frames.

**Proposition 4.4.**  $krb1 \sqsubseteq_{\pi_{11}^{-1}, \pi_{11}} kt1in$ .

The mediators  $\pi_{a01}^{ir}$  and  $\pi_{a01}^{ri}$  and associated simulation relations for refining  $a0i$  and  $a0n$  are defined analogously to Section 4.4.2. The authentication guards can be defined or discovered as described in that section. The replay cache guarantees injective agreement with the initiator to the responder, while the initiator obtains only a non-injective agreement with the responder. The proof of injectivity in the refinement of  $commit_{a0i}$  by  $step5_{krb1}$  requires an invariant stating that if a responder  $B$  knowing a key  $Kab$  and a timestamp  $Ta$  then he has an entry  $(B, Kab, Ta)$  in the replay cache during  $Ta$ 's validity. Appendix A provides some proof details.

**Proposition 4.5.** (i)  $krb1 \sqsubseteq_{\pi_{a01}^{ir}} a0n$  for the initiator and (ii)  $krb1 \sqsubseteq_{\pi_{a01}^{ri}} a0i$  for the responder.

#### 4.4.4 Key freshness

Finally, we formalize key freshness (**R5**) for the responder as an invariant of  $krb1$ . This property expresses that a session key  $K$  appearing in a run of a responder  $B$  with an initiator  $A$  uniquely identifies that run, provided that  $A$  and  $B$  are honest.

$$\begin{aligned} rfresh_{krb1} \equiv & \{s \mid \forall R, R', A, A', B, B', K, Ts, Ts', Ta, Ta' . \\ & s.runs(R) = (Resp, A, B, [K, Ts, Ta]) \wedge \\ & s.runs(R') = (Resp, A', B', [K, Ts', Ta']) \wedge \\ & B \notin bad \wedge A \notin bad \rightarrow R = R' \} \end{aligned}$$

We have proved that this is an external invariant of  $krb1$ .

**Proposition 4.6.**  $oreach(krb1) \subseteq rfresh_{krb1}$ .

All cases except for the critical Step 5 by the responder are proved automatically.<sup>3</sup> In the latter case, the proof relies on other invariants proved for  $krb1$  or inherited from its ancestors. Secrecy and the responder's agreement with the server are used to exclude the cases where the initiator  $A'$  or the responder  $B'$  in the run  $R'$  is dishonest. Otherwise, we use the injective agreement with the initiator to reduce the proof to the initiator's key freshness (proved for  $ktlin$ ).

## 4.5 Channel Protocols (L2)

As described in Section 3.5, at Level 2, we extend the state with a field  $chan$  for the set of channel messages of type  $chmsg$ . All fields except  $chan$  are observable, i.e., the observation function  $\pi_{runs,clk,cch}$  projects  $\Sigma_{krb2}$  to  $\Sigma_{krb1}$ .

$$\Sigma_{krb2} \triangleq \Sigma_{krb1} + (\mid chan \in \mathcal{P}(chmsg) \mid)$$

For the refinement of  $krb1$ , we use the simulation relation  $R_{12} \equiv \Pi_{runs,clk,cch}$  with the identity mediator function. In the protocol events, we add message-receiving guards, which replace the authorization and authentication guards from Level 1 (if any), and

<sup>3</sup>**FIX:** This paragraph may need a minor revision. I will go through the proof again, also for answering your question related to the slides.

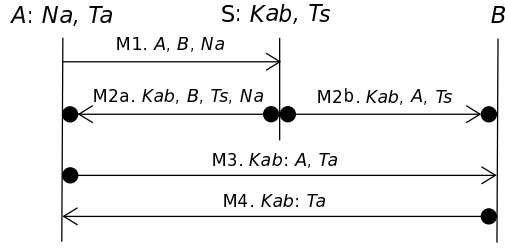


Figure 11: Channel-based Kerberos protocol (*krb2*)

actions for sending channel messages. The local guards remain the same. Moreover, we introduce an active intruder as described in Section 3.5. The intruder’s *fake* event refines *skip*, as it only modifies *chan*, while all other events refine their counterparts in the model *krb1*.

The channel-based refinement *krb2* of *krb1* is shown in Figure 11. The protocol is now started by the initiator sending *A, B* together with his nonce *Na* to the server. The server uses static secure channels to send the session key *Kab*, the name *B*, the timestamp *Ts*, and the nonce *Na* to the initiator *A* and *Kab, A, Ts* to the responder *B*. The responder *B* obtains key confirmation from *A* by receiving *A, Ta* on a dynamic authentic channel protected by the session key *Kab* (and similarly for *A*’s guarantee in the other direction). No confidentiality is required in the key confirmation phase.

As example events, we describe the changes in Steps 3 and 5. In the server’s Step 3, we add an additional guard for receiving message M1 and an action for sending messages M2a and M2b.

$$\begin{aligned} \text{Insec}([A, B, Na]) \in s.chan \quad & \text{-- receive M1} \\ s'.chan := s.chan \cup \{ & \text{Secure}(S, A, [Kab, B, Ts, Na]), \text{Secure}(S, B, [Kab, A, Ts]) \} \end{aligned}$$

In Step 5, the responder *B* receives message M2b from the server and M3 from the initiator *A* and sends message M4. Here, the message-receiving guards replace the previous authorization and authentication guards.

$$\begin{aligned} \text{Secure}(S, B, [Kab, A, Ts]) \in s.chan \\ \text{dAuth}(Kab, [A, Ta]) \in s.chan \\ s'.chan := s.chan \cup \{ \text{dAuth}(Kab, [Ta]) \} \end{aligned}$$

The refinement proof requires additional invariants. Many of these are directly suggested by the guard strengthening proof obligations stating that the message-receiving guards (L2) imply the security guards (L1). This is one of the main benefits of using guard protocols as a link between the properties and the message-based protocols. Appendix B contains a proof sketch of guard strengthening for Step 5.

**Proposition 4.7.** *Let  $I_{krb2}$  be the intersection of the invariants of *krb2* and let  $R'_{12} \equiv R_{12} \cap (\Sigma_{krb1} \times I_{krb2})$ . Then  $\text{reach}(krb2) \subseteq I_{krb2}$  and  $krb2 \sqsubseteq_{R'_{12}, id} krb1$ .*

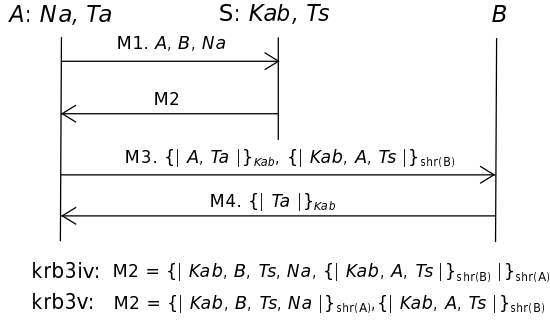


Figure 12: Cryptographic core Kerberos protocols

## 4.6 Cryptographic Protocols (L3)

As described in Section 3.6, in our setup at Level 3, each agent  $A$  shares a long-term symmetric key  $shr(A)$  with the server  $S$ . We concretize the initial key setup relation by defining  $keySetup \equiv \{(shr(A), C) \mid C = A \vee C = S\}$ , thereby establishing **(A3)**. In the state, we replace the set of channel messages,  $chan$ , by a set of cryptographic messages,  $IK$  (for intruder knowledge). All fields except  $IK$  are observable. Initially,  $IK$  holds the corrupted long-term keys, i.e.,  $shr(bad)$ .

$$\Sigma_{krb3v} \triangleq \Sigma_{krb1} + (\!| IK \in \mathcal{P}(msg) |\!)$$

The simulation relation  $R_{23}$  with  $krb2$  is defined as the intersection of the relations  $R_{23}^{msgs}$ ,  $R_{23}^{key}$ , and  $R_{23}^{non}$ , from Section 3.6 with  $\Pi_{runs,clk,cch}$ . The relation  $R_{23}^{msgs}$  is parametrized by the protocol-dependent message abstraction function  $absMsg \in \mathcal{P}(msg) \rightarrow \mathcal{P}(chmsg)$ , which we will instantiate to the Kerberos protocols below.

By refining the channel-based intruder into a standard Dolev-Yao intruder, as described in Section 3.6, we also establish **(A1)**. In the protocol events, we replace the channel messages by cryptographic ones. In general, there are alternative realizations using different cryptographic operations.

Figure 12 shows the core of Kerberos 4 [51] and Kerberos 5 [45]. We implement the static secure channels by encryption with the long-term keys and we refine the dynamic authentic channels into encryptions with session keys. (In the Dolev-Yao model, symmetric encryption also provides authenticity.)

We also modify the communication topology of the channel-based model: The initiator now relays the responder's ticket from the server. While in Kerberos 5 the ticket is sent alongside the ciphertext containing the initiator's session key, it appears inside the ciphertext in Kerberos 4. We now present our models of Kerberos 5 and Kerberos 4,  $krb3v$  and  $krb3iv$ .

### 4.6.1 Kerberos 5 protocol (L3)

We focus on the refinement of Steps 3–5, which reflect the modified communication topology. In Step 3, the server sends the message M2 by adding it to  $IK$ . This message

consists of a pair of ciphertexts and replaces the messages M2a and M2b in *krb2*. In Step 4, the initiator *A* receives M2 and forwards its second component along with the authenticator  $\{\!\{ A, Ta \}\!\}_{Kab}$ , which proves that *A* knows *Kab*.

$$\begin{aligned} \{\!\{ Kab, B, Ts, Na \}\!\}_{shr(A)}, X \} &\in s.IK && \text{-- recv M2} \\ s'.IK := s.IK \cup \{\!\{ \{\!\{ A, Ta \}\!\}_{Kab}, X \}\!\} &&& \text{-- send M3} \end{aligned}$$

The responder receives the two-component message M3 in Step 5 and sends back the confirmation message M4.

$$\begin{aligned} \{\!\{ \{\!\{ A, Ta \}\!\}_{Kab}, \{\!\{ Kab, A, Ts \}\!\}_{shr(B)} \}\!\} &\in s.IK && \text{-- recv M3} \\ s'.IK := s.IK \cup \{\!\{ \{\!\{ Ta \}\!\}_{Kab} \}\!\} &&& \text{-- send M4} \end{aligned}$$

The message abstraction function, *absMsg*, abstracts the components of messages M2 and M3 separately. For instance, here are the rules defining the abstraction of the initiator's encrypted key, the responder tickets, and the authenticators.

$$\frac{\{\!\{ K, B, T, N \}\!\}_{shr(A)} \in H}{\text{Secure}(S, A, [K, B, T, N]) \in \text{absMsg}(H)} \quad \frac{\{\!\{ K, A, T \}\!\}_{shr(B)} \in H}{\text{Secure}(S, B, [K, A, T]) \in \text{absMsg}(H)}$$

$$\frac{\{\!\{ A, T \}\!\}_K \in H}{\text{dAuth}(K, [A, T]) \in \text{absMsg}(H)}$$

Here, the possibility to abstract message parts, which is reflected in the definition of the simulation relation as  $\text{absMsg}(\text{parts}(s.IK)) \subseteq t.\text{chan}$  (cf. Section 3.6.3), allows us to abstract each component of message M2 separately. This enables the modification of the communication topology between Levels 2 and 3.

The refinement proof requires only four simple additional invariants: Two concern key definedness and two state that the long-term keys the intruder knows are exactly those of the dishonest agents (Assumption A2). We have already established all other relevant properties on higher levels of abstraction.

**Proposition 4.8.** *Let  $I_{krb3v}$  be the intersection of the invariants of *krb3v* and let  $R_{23}^v \equiv R_{23} \cap (\Sigma_{krb2} \times I_{krb3v})$ . Then we have  $\text{reach}(krb3v) \subseteq I_{krb3v}$  and  $krb3v \sqsubseteq_{R_{23}^v, id} krb2$ .*

#### 4.6.2 The Kerberos 4 protocol (L3)

In the core Kerberos 4 protocol, the responder's ticket is encrypted inside the initiator's message from the server. Hence, message M2 is modified as follows.

$$M2'. S \rightarrow A : \{\!\{ Kab, B, Ts, Na, \{\!\{ Kab, A, Ts \}\!\}_{shr(B)} \}\!\}_{shr(A)}$$

The changes in the model are straightforward. Moreover, the simulation relation,  $R_{23}'$ , is the same as  $R_{23}$  above except for a minor change in the message abstraction function: the cryptographic messages abstracted to message M2a, i.e.,  $\text{Secure}(S, A, [K, B, T, N])$  (see Figure 11), are now of the form  $\{\!\{ K, B, T, N, X \}\!\}_{shr(A)}$  and include a message variable *X* for the responder's ticket instead of  $\{\!\{ K, B, T, N \}\!\}_{shr(A)}$  for Kerberos 5.

The refinement proof for *krb3iv* requires additional invariants. One invariant links the variable *X* above with the ticket by describing the ticket's shape and the encrypted key *K* in message M2 received by an honest initiator *A*.

$$\begin{aligned}
ticket_{krb3iv} &\equiv \{s \mid \forall A, B, T, N, K, X. \\
&\llbracket K, B, T, N, X \rrbracket_{shr(A)} \in parts(s.IK) \wedge A \notin bad \rightarrow \\
&X = \llbracket K, A, T \rrbracket_{shr(B)} \wedge K \notin ran(shr) \}
\end{aligned}$$

Another invariant expresses that the addition of session keys to the intruder's knowledge does not reveal additional keys.

$$\begin{aligned}
sessK_{krb3iv} &\equiv \{s \mid \forall KS, K. \\
&KS \subseteq key \setminus ran(shr) \rightarrow (K \in analz(KS \cup s.IK) \leftrightarrow K \in KS \vee K \in analz(s.IK)) \}
\end{aligned}$$

A similar invariant states that the intruder cannot derive new nonces by learning new session keys. We then prove:

**Proposition 4.9.** *Let  $I_{krb3iv}$  be the intersection of the invariants of  $krb3iv$  and let  $R_{23}^{iv} \equiv R'_{23} \cap (\Sigma_{krb2} \times I_{krb3iv})$ . Then  $reach(krb3iv) \subseteq I_{krb3iv}$  and  $krb3iv \sqsubseteq_{R_{23}^{iv}, id} krb2$ .*

## 4.7 Discussion

### 4.7.1 Overall security guarantees at Level 3

Having gone through a number of refinements, the reader may wonder at this point what is the precise relationship between the secrecy and authentication properties proved as invariants at Level 0 and the final models at Level 3. The answer is given by Proposition 2.6 and Corollary 2.8, which state that refinement is transitive and that *external* invariants (such as the secrecy and agreement invariants of  $s0$ ,  $a0n$ , and  $a0i$ ) are preserved along refinements. The mediator function translates the observations from concrete to abstract models.

As an example, consider the series of refinements in Figure 6: we have refined the secrecy model  $s0$  in five steps into the core Kerberos 5 model ( $krb3v$ ). The composed mediator function along the right-hand side of this figure is  $\pi_{s01} \circ \pi_{1in1} \circ \pi_{11}$ . The first part,  $\pi_{1in1} \circ \pi_{11}$ , projects the state of the model  $krb3v$  with fields  $runs$ ,  $clk$ ,  $cch$ , and  $IK$  to the state of the model  $kt1$ , which has only the  $runs$  variable. Moreover, using the notation of Figure 6, the frames of the initiator, responder, and server runs are projected from  $([Kab, Ts, Ta], [Kab, Ts, Ta, END], [Na, Ts])$  to  $([Kab], [Kab], [])$ , thereby removing everything but the session key  $Kab$ .

The second part, the mediator function  $\pi_{s01}$  (defined in Section 4.4.1), transforms this minimal state information to the knowledge and authorization relations  $kn$  and  $az$  of the model  $s0$ . Hence, using Proposition 2.4, the following overall secrecy result can be derived as a combination of Propositions 3.1, 4.1, 4.3, 4.4, 4.7, and 4.8.

**Corollary 4.10.**  $(\pi_{s01} \circ \pi_{1in1} \circ \pi_{11})(oreach(krb3v)) \subseteq secrecy$ .

In a similar way, we can express the authentication results directly as properties of  $krb3v$  (cf. Figure 7). The initiator's injective agreement with the server on  $(Kab, B, Na, Ts)$  and his non-injective agreement on  $(Kab, Ts, Ta)$  with the responder are summarized in the following corollary.

**Corollary 4.11.** *For the initiator, we have*

	definitions	lemmas	lines	elapsed	cpu	factor
infrastructure	54	326	3343	5	12	2.4
Level 1 models	47	138	2023	18	44	2.4
Kerberos 5	73	184	2996	58	221	3.8
NSSK	73	202	3357	40	149	3.7
Denning-Sacco	56	96	1896	19	68	3.6

Table 4: Overall specification and proof statistics. Times are indicated in seconds.

1.  $(\pi_{a01}^{is} \circ \pi_{1in1} \circ \pi_{11})(oreach(krb3v)) \subseteq iagree$ , and
2.  $(\pi_{a01}^{ir} \circ \pi_{11})(oreach(krb3v)) \subseteq niagree$ .

We complete the picture by stating the authentication guarantees for the responder with the server and with the initiator.

**Corollary 4.12.** *For the initiator, we have*

1.  $(\pi_{a01}^{rs} \circ \pi_{1in1} \circ \pi_{11})(oreach(krb3v)) \subseteq niagree$ , and
2.  $(\pi_{a01}^{ri} \circ \pi_{11})(oreach(krb3v)) \subseteq iagree$ .

Summarizing, given a security property  $P$  proved as an external invariant of a model  $S$  (e.g., at Level 0) and a series of refinements of  $S$  into a model  $S'$  (e.g., at Level 3) with composed mediator function  $\pi$ , Proposition 2.4 yields the guarantee  $\pi(oreach(S')) \subseteq P$  for  $S'$ , i.e.,  $\pi$  transforms the set of concrete observations of  $S'$  to a set of abstract observations satisfying  $P$ .

## 4.7.2 Specification and proof statistics

We have summarized some statistics from our developments in Table 4, which has entries for four groups of theories. For each group we indicate the number of definitions and lemmas that we formalized in Isabelle/HOL, the number of lines of the corresponding theory files, the elapsed and cpu times for proof checking,<sup>4</sup> and the speedup factor. The measurements were made on a 2.6 GHz Intel Core i7 (quad-core) laptop with 8 GB RAM running Isabelle/HOL 2013 configured to work with 4 threads.

The first group consists of the infrastructure theories, which support our method, and includes the general theory of refinement (Section 2.2) as well as the specific infrastructure for security protocol modeling and refinement including the Level-0 models of secrecy and authentication (Section 3). The second group is formed by the Level-1 models *kt1*, *kt1in*, and *kt1nn* (Sections 4.4.1 and 4.4.2). The third, fourth, and fifth lines list the data for the models *krb1*, *krb2*, and *krb3v* pertaining to the Kerberos 5 protocol (Sections 4.4.3, 4.5, and 4.6.1), the models *nssk1*, *nssk2*, and *nssk3* related to the Needham-Schroeder Shared-Key (NSSK) protocol, and the models *ds1*, *ds2*, and *ds3* for the Denning-Sacco protocol (see also Figure 5).

<sup>4</sup>The times are proof checking only without session image or document generation.

level	model	lemmas	invariants	refines
L1	<i>kt1</i>	30	1 / 1	<i>s0</i>
	<i>kt1in</i>	58	2 / 2	<i>a0i, a0n</i>
	<i>kt1nn</i>	50	1 / 0	<i>a0n</i> (twice)
	<i>krb1</i>	77	3 / 5	<i>kt1in, a0i, a0n</i>
	<i>nssk1</i>	71	2 / 5	<i>kt1in, a0n</i> (twice)
L2	<i>krb2</i>	67	9 / 4	<i>krb1</i>
	<i>nssk2</i>	69	10 / 4	<i>nssk1</i>
L3	<i>krb3iv</i>	59	7 / 0	<i>krb2</i>
	<i>krb3v</i>	32	4 / 0	<i>krb2</i>
	<i>nssk3</i>	62	9 / 0	<i>nssk2</i>

Table 5: Detailed proof statistics: lemmas, new / inherited invariants, and refinements

Table 5 shows more detailed proof statistics for the models used in the development of the Kerberos 4 and 5 and the NSSK protocols. In particular, the fourth column lists two numbers for invariants. The first number denotes *new* invariants, which are proved by induction and primarily used to strengthen the simulation relation in refinement proofs. One type of (internal) invariant, which we find in most models at all levels, are key definedness invariants. These relate session keys to the domain of the variable *runs*; they are easy to prove and serve mainly a technical purpose. These invariants are therefore not further mentioned below. The second number indicates invariants that are *inherited* from higher-level models. This number does not include all inherited invariants, but only those that are needed in a proof at the same or a lower level. For example, the abstract Kerberos and NSSK models, *krb1* and *nssk1*, inherit the same five invariants from their common ancestor models. Hence, these invariants can be used in further proofs without the need to reestablish them.

As mentioned earlier, all system requirements are already established at Level 1, mainly in the form of refinements of Level-0 models. Exceptions are key freshness properties, which are formulated as invariants. There is one other new invariant at Level 1, which states a property of the cache in the abstract Kerberos model *krb1*.

The largest number of invariants is required at Level 2. These invariants can be classified into two groups: those concerning session key secrecy (2 each for *krb2* and *nssk2*) and those relating received messages to their sender’s or receiver’s state (5 for *krb2* and 6 for *nssk2*). Many of these invariants arise naturally from guard refinement proof obligations, e.g., to establish that the received messages imply an authorization or authentication guard at Level 1. All but two of them have short straightforward proofs (2-6 lines of proof script).

The invariants remaining at Level 3 mainly concern details introduced at that level. Examples are the secrecy of long-term keys, an invariant about the shape of forwarded

tickets that are encrypted with an honest agent’s long-term key ( $krb3iv$  and  $nssk3$ ), and an invariant about the non-use of session keys to encrypt other session keys or nonces ( $krb3iv$  and  $nssk3$ ).

## 5 Related Work

There have been other proposals for developing security protocols by refinement using various formalisms such as the B method [16], its combination with CSP [22], Event-B [15], I/O automata [35], and ASMs [14]. None of these continue their refinements to the level of a full Dolev-Yao intruder. Either they only consider an intruder that is passive [35], defined ad-hoc [22, 14, 15], or that corresponds to our Level-2 intruder [16]. This makes a comparison of their results with standard protocol models difficult. Moreover, these works do not propose a uniform and systematic development method as we do with our four-level refinement strategy and most of them develop individual protocols rather than entire families.

Datta et al. [27] use protocol templates with messages containing function variables to specify and prove properties of protocol classes. Refinement here means instantiating function variables and discharging the associated assumptions. Based on the same notion of refinement, Pavlovic et al. [48, 24] propose specialized logics for proving interdependent secrecy and authentication properties. In contrast to our approach, these refinements do not involve a fundamental change of the abstraction level since one abstracts and instantiates operations on messages. Moreover, the soundness of these refinements is not formally justified.

Abstract channels and their transformations were studied by Maurer and Schmid [38]. Boyd has formalized analogous results using Z [19]. Bieber et al. [17] model abstract channels using the B method and refine them to cryptographic implementations. Bella et al. [11] combine messages transmitted over channels with security properties with cryptographic messages in order to model “second-level” protocols, which rely on the services of some underlying protocols (such as a secure SSL channel). Abadi et al. [1] formalize secure channels in a variant of the join calculus and establish full abstraction results for translations to cryptographic implementations.

Mödersheim and Viganò [43] propose a compositional approach, where protocol specifications may combine channel messages and cryptographic messages. Channel messages can later be replaced by protocols realizing their properties. Their objective is to identify the weakest condition that allows protocols to securely implement channels. In contrast, we use refinement to transform channel protocols into concrete cryptographic protocols.

Classical notions of refinement (such as simulation) do not preserve information-flow properties, since they involve a reduction of non-determinism, which can destroy secrecy. Several works address this problem, known as the refinement paradox, for example, [37, 5, 32, 39]. Morgan and McIver [39, 41, 42] solve the paradox by explicitly recording the set of possible values of secret variables. These sets represent the intruder’s ignorance and refinements may extend, but never reduce them. Cortier et al. [25] show that strong secrecy is equivalent to reachability-based secrecy (used here), if the secrets are not tested. Key establishment protocols that include a key

confirmation phase (such as such as Kerberos and NSSK) do not satisfy this condition.

Simulation-based security [23, 7] is a paradigm for specifying idealized functionalities and implementing them using a notion of secure emulation. Delaune et al. [28] have recently proposed a symbolic version of this paradigm, which can be understood as a form of compositional refinement. The compositionality comes at the price of requiring an public-key encryption/decryption functionality and proving a joint-state theorem. As an example they derive the Needham-Schroeder-Lowe protocol. The modeling of encryption as a service considerably reduces the abstraction level of these models compared to the standard symbolic representation as a term constructor.

Our most concrete models are still quite abstract when compared to a protocol implementation in a programming language such as C or Java. The work by Polikarpova and Moskal [49] can be seen as an extension of our refinement levels with two additional levels towards an implementation: one in which the messages are replaced by bitstrings and one that represents the real implementation. The refinements are encoded and proved in the general-purpose C program verifier VCC.

Several other researchers have analyzed Kerberos. Bella and Riccobene [14] develop Kerberos 4 in three refinements using ASMs. They use a non-standard attacker model and prove mostly liveness properties (e.g., all runs reach a specific state) instead of secrecy and authentication properties. Bella and Paulson model BAN Kerberos [13], Kerberos 4 [12], and Kerberos 5 [10] including session key compromise using the inductive approach [47]. However, they do not model a replay cache and prove only non-injective agreements.

Butler et al. [21, 20] constructed detailed models of Kerberos 5 using multiset rewriting. These models include cross-realm authentication and other realistic features such as options, flags, and error handling. They manually construct their models and proofs in several “refinements” to keep them manageable. However, their notion of refinement is informal.

## 6 Conclusions

Our development provides strong evidence that refinement supports the systematic understanding and development of families of protocols. The abstract models help the developer to focus on the essentials: In our case studies, we have established all requirements on guard protocol models (L1), which contain neither messages, communication channels, nor intruder events. Our refinement strategy guides the developer towards the concrete levels that account for the environment assumptions, namely, the distributed environment controlled by a Dolev-Yao intruder. The abstraction levels of our refinement strategy are reflected in well-structured proofs of correctness, where the simulation relations used are either fixed (a projection at L1-L2) or systematically derived (e.g., abstraction of runs to signals at L0-L1 and cryptographic to channel messages at L2-L3). Our case studies also show that our development strategy and tools scale to realistic protocols with non-trivial features.

A central part of this work has been the development and exploitation of guard protocols, which form the bridge between security properties and channel protocols, i.e., from the “what” to the “how”. Security guards realize properties abstractly. Moreover,

they substantially simplify proof construction. They give rise to invariants in a canonical way during refinement, thereby facilitating invariant discovery. These invariants strengthen the simulation relations in the refinement proofs.

Our channel protocol model is quite simple and protocol messages with nested cryptographic operations or undecryptable message parts have no direct representation. This excludes modeling, for example, messages containing certificates, the forwarding of undecryptable messages, and nested encryption (NSSK and Kerberos). Our experience has convinced us that this simplicity is a virtue rather than a limitation. Our models of server-based key transport protocols naturally reflect their actual (star-shaped) security architecture. We view forwarding and double encryption as implementation techniques, to be dealt with at the final level. Our developments show that this is possible. Such abstractions are even more beneficial for developing new protocols. From this perspective, certificates provide an abstract authentic channel from the certification authority to the agent verifying the certificate's content and encrypted and signed messages are just one way of implementing a secure channel.

**Future Work** We ultimately envision a tool-based development process where engineers can choose standard properties and follow high-level recipes for building guard, channel, and crypto protocols, with tools checking their steps along the way. To achieve this, we will work into two directions. First, we want to extend the range of protocols that can be modeled and reasoned about. For example, we plan to add support for Diffie-Hellman key agreement, compromising adversaries, and more complex properties such as perfect forward secrecy, possibly along the lines of [9]. Second, we would like to automate development based on our strategy. It should be possible to derive protocol models directly from high-level descriptions such as the authentication graphs of Figure 7 and sequence charts of Figures 8–10. Moreover, with suitable infrastructure it should be feasible to automatically generate and (as far as possible) prove invariants and simulations, given their strong regularity.

## Acknowledgements

This work is partially supported by the EU FP7-ICT-2009.1.4 Project No. 256980, NESSoS: Network of Excellence on Engineering Secure Future Internet Software Services and Systems. We are grateful to Jean-Raymond Abrial for fruitful discussions in the early stages of this work. We would also like to thank Ivano Somaini for developing parts of the Isabelle/HOL theories and Martin Vechev, Vincent Jugé, Son Thai Hoang, Eugen Zălinescu, Binh Thanh Nguyen, and Ognjen Maric for their helpful comments on earlier drafts.

## References

- [1] M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. *Inf. Comput.*, 174(1):37–83, 2002.

- [2] M. Abadi and L. Lamport. The existence of refinement mappings. *Theor. Comput. Sci.*, 82(2):253–284, 1991.
- [3] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
- [4] J.-R. Abrial and S. Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundam. Inform.*, 77(1-2):1–28, 2007.
- [5] R. Alur, P. Cerný, and S. Zdancewic. Preserving secrecy under refinement. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Proc. 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, number 4052 in Lecture Notes in Computer Science, pages 107–118, 2006.
- [6] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, 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 K. Etessami and S. K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.
- [7] M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Inf. Comput.*, 205(12):1685–1720, 2007.
- [8] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, June 2005.
- [9] D. A. Basin and C. J. F. Cremers. Modeling and analyzing security in the presence of compromising adversaries. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 340–356. Springer, 2010.
- [10] G. Bella. *Formal Correctness of Security Protocols*. Information Security and Cryptography. Springer, 2007.
- [11] G. Bella, C. Longo, and L. C. Paulson. Verifying second-level security protocols. In D. A. Basin and B. Wolff, editors, *TPHOLs*, volume 2758 of *Lecture Notes in Computer Science*, pages 352–366. Springer, 2003.
- [12] G. Bella and L. C. Paulson. Kerberos version 4: Inductive analysis of the secrecy goals. In *Proc. 5th European Symposium on Research in Computer Security (ESORICS)*, pages 361–375, 1998.
- [13] G. Bella and L. C. Paulson. Mechanising BAN Kerberos by the inductive method. In A. J. Hu and M. Y. Vardi, editors, *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1998.
- [14] G. Bella and E. Riccobene. Formal analysis of the Kerberos authentication system. *Journal of Universal Computer Science*, 3(12):1337–1381, 1997.

- [15] N. Benaïssa. *La composition des protocoles de sécurité avec la méthode B événementielle*. PhD thesis, Université Henri Poincaré - Nancy I, France, May 2010. (In French).
- [16] P. Bieber and N. Boulahia-Cuppens. Formal development of authentication protocols. In *Sixth BCS-FACS Refinement Workshop*, 1994.
- [17] P. Bieber, N. Boulahia-Cuppens, T. Lehmann, and E. van Wickeren. Abstract machines for communication security. In *Proc. 6th IEEE Computer Security Foundations Workshop (CSFW)*, pages 137–146, 1993.
- [18] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001.
- [19] C. Boyd. Security architectures using formal methods. *IEEE Journal on Selected Areas in Communications*, 11(5), 1993.
- [20] F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. A formal analysis of some properties of Kerberos 5 using MSR. In *Proc. 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 175–. IEEE Computer Society, 2002.
- [21] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, and C. Walstad. Formal analysis of Kerberos 5. *Theoretical Computer Science*, 367:57–87, November 2006.
- [22] M. J. Butler. On the use of data refinement in the development of secure communications systems. *Formal Aspects of Computing*, 14(1):2–34, 2002.
- [23] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
- [24] I. Cervesato, C. Meadows, and D. Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations*, pages 48–61, Washington, DC, USA, 2005.
- [25] V. Cortier, M. Rusinowitch, and E. Zalinescu. Relating two standard notions of secrecy. *Logical Methods in Computer Science*, 3(3), 2007.
- [26] C. J. F. Cremers. The Scyther tool: Verification, falsification, and analysis of security protocols. In A. Gupta and S. Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.
- [27] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositionl logic for security protocols. *Journal of Computer Security*, 13:423–482, 2005.
- [28] S. Delaune, S. Kremer, and O. Pereira. Simulation based security in the applied pi calculus. In R. Kannan and K. N. Kumar, editors, *FSTTCS*, volume 4 of *LIPICs*, pages 169–180. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.

- [29] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.
- [30] L. Gong. Variations on the themes of message freshness and replay - or the difficulty in devising formal methods to analyze cryptographic protocols. In *In Proceedings of the Computer Security Foundations Workshop VI (CSFW)*, pages 131–136, 1993.
- [31] ISO. Information Technology – Security Techniques – Entity Authentication Mechanisms – Part 3: Entity Authentication Using a Public-key Algorithm ISO/IEC 9798-3. International Standard, 2nd edition, 1998.
- [32] J. Jürjens. Secrecy-preserving refinement. In *Proc. 10th Symposium on Formal Methods Europe (FME 2001)*, number 2021 in Lecture Notes in Computer Science, pages 135–152. Springer, 2001.
- [33] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software — Concepts and Tools*, 17:93–102, 1996.
- [34] G. Lowe. A hierarchy of authentication specifications. In *IEEE Computer Security Foundations Workshop*, pages 31–43, Los Alamitos, CA, USA, 1997. IEEE Computer Society.
- [35] N. A. Lynch. I/O automaton models and proofs for shared-key communication systems. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, pages 14–29, 1999.
- [36] N. A. Lynch and F. W. Vaandrager. Forward and backward simulations: I. untimed systems. *Inf. Comput.*, 121(2):214–233, 1995.
- [37] H. Mantel. Preserving information flow properties under refinement. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 78–91, 2001.
- [38] U. M. Maurer and P. E. Schmid. A calculus for secure channel establishment in open networks. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS)*, pages 175–192, 1994.
- [39] A. McIver and C. C. Morgan. Sums and lovers: Case studies in security, compositionality and refinement. In *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, pages 289–304, 2009.
- [40] R. Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.
- [41] C. Morgan. The shadow knows: Refinement of ignorance in sequential programs. In *Mathematics of Program Construction, 8th International Conference, MPC 2006, Kuressaare, Estonia, July 3-5, 2006, Proceedings*, volume 4014 of LNCS, pages 359–378, 2006.

- [42] C. Morgan. The shadow knows: Refinement and security in sequential programs. *Science of Computer Programming*, 74(8):629–653, 2009.
- [43] S. Mödersheim and L. Viganò. Secure pseudonymous channels. In M. Backes and P. Ning, editors, *Proc. 14th European Symposium on Research in Computer Security (ESORICS)*, volume 5789 of *Lecture Notes in Computer Science*, pages 337–354. Springer, 2009.
- [44] R. Needham and M. D. Schroeder. Using encryption for authentication in large data networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
- [45] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, 1994.
- [46] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.
- [47] L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.
- [48] D. Pavlovic and C. Meadows. Deriving secrecy in key establishment protocols. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*, pages 384–403, 2006.
- [49] N. Polikarpova and M. Moskal. Verifying implementations of security protocols by refinement. In R. Joshi, P. Müller, and A. Podelski, editors, *VSTTE*, volume 7152 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2012.
- [50] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF)*, pages 78–94, 2012.
- [51] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Winter 1988 Usenix Conference*, Feb. 1988.

## A Proof of Step 5 in Proposition 4.5(ii) (L1)

We sketch the proof of guard strengthening in the refinement of the abstract event  $\text{commit}_{a0i}([B, A], (Kab, Ts, Ta))$  by the concrete  $\text{step5}_{krb1}(Rb, A, B, Kab, Ts, Ta)$  of the responder  $B$ , which is part of the proof that establishes the responder's injective agreement with the initiator. We define the function  $\text{sigC}(r) \equiv m_r$  that reconstructs the abstract signals from the runs  $r$ , where  $m_r$  is defined as follows.

$$\begin{aligned} m_r(\text{Running}([B, A], (Kab, Ts, Ta))) &\equiv c_I \\ m_r(\text{Commit}([B, A], (Kab, Ts, Ta))) &\equiv c_R \end{aligned}$$

Here,  $c_I$  and  $c_R$  are the following cardinalities.

$$\begin{aligned} c_I &\equiv |\{Ra \mid \exists nl. r(Ra) = (\text{Init}, A, B, Kab\#Ts\#Ta\#nl)\}| \\ c_R &\equiv |\{Rb \mid \exists nl. r(Rb) = (\text{Resp}, A, B, Kab\#Ts\#Ta\#nl)\}| \end{aligned}$$

For the remaining cases, we set  $m_r(x) \equiv 0$ . The guard strengthening proof obligation requires that we prove

$$c_R < c_I, \quad (3)$$

assuming that  $A$  and  $B$  are honest and that the guards of  $\text{step5}_{krb1}$  are satisfied. Since we use a cache to prevent a responder  $B$  from accepting the same key  $Kab$  and timestamp  $Ta$  multiple times, there cannot be any prior execution of Step 5 with these parameters. We therefore strengthen the subgoal (3) to the conjunction of the following two subgoals.

$$c_I > 0 \quad (4)$$

$$c_R = 0 \quad (5)$$

Subgoal (4) follows from the authentication guard for  $\text{step5}_{krb1}$ . Subgoal (5) requires that there is no responder run  $Rb'$  and list  $nl'$  corresponding to a commit signal, i.e.,

$$s.\text{runs}(Rb') = (\text{Resp}, A, B, Kab\#Ts\#Ta\#nl'). \quad (6)$$

We prove this by contradiction, using an invariant stating that (6) and the validity of  $Ta$  (i.e.,  $s.\text{clk} < Ta + La$ ) entail the existence of a cache entry  $(B, Kab, Ta) \in s.\text{cch}$ . This entry was added in a previous execution of Step 5 by run  $Rb'$  and is not yet purged from the cache, since the timestamp  $Ta$  is still valid. Hence, by assuming (6) and noting that  $s.\text{clk} < Ta + La$  and the replay check  $(B, Kab, Ta) \notin s.\text{cch}$  are guards of Step 5, we use the invariant to derive a contradiction. This concludes the proof that the guards of the  $\text{step5}_{krb1}$  event imply the guard of  $\text{commit}_{a0i}$ .

## B Proof of Step 5 in Proposition 4.7 (L2)

The guard strengthening proof obligation for Step 5 requires that the guards for receiving M2b and M3 in the responder  $B$ 's  $\text{step5}_{krb2}$  imply the following three non-local security guards of  $\text{step5}_{krb1}$ : the authorization guard

$$(Kab, B) \in azC(s.\text{runs}), \quad (7)$$

the server authentication guard

$$B \notin bad \rightarrow \exists Rs, Na. Kab = Rs\$sk \wedge s.runs(Rs) = (Serv, A, B, [Na, Ts]) \quad (8)$$

and the initiator authentication guard

$$A \notin bad \wedge B \notin bad \rightarrow \exists Ra, nl. s.runs(Ra) = (Init, A, B, Kab\#Ts\#Ta\#nl). \quad (9)$$

The following invariant directly arises from the proof obligation expressing the strengthening of the server authentication guard (8). It expresses the guarantee that an honest  $B$  gets about the server's state from receiving message M2b.

$$\begin{aligned} M2b_{krb2}^+ &\equiv \{s \mid \forall Kab, A, B, Ts. \\ &\text{Secure}(S, B, [Kab, A, Ts]) \in s.chan \wedge B \notin bad \rightarrow \\ &\exists Rs, Na. Kab = Rs\$sk \wedge s.runs(Rs) = (Serv, A, B, [Na, Ts]) \} \end{aligned}$$

A related invariant,  $M2b_{krb2}^-$ , describes the case where  $B$  is dishonest. In this case,  $Kab$  is either a corrupted key or a session key generated by the server for some dishonest agent. These two invariants suffice to derive the strengthening of the authorization guard (7).

The strengthening of the initiator authentication guard (9) corresponds to the following proof obligation. It describes the authentication guarantee that the responder  $B$  gets about the initiator's state from receiving messages M2b and M3:  $A$  knows  $Kab$ ,  $Ts$ , and  $Ta$  in an initiator run  $Na$ , provided that  $A$  and  $B$  are honest.

$$\begin{aligned} &\text{Secure}(S, B, [Kab, A, Ts]) \in s.chan \wedge dAuth(Kab, [A, Ta]) \in s.chan \wedge \\ &A \notin bad \wedge B \notin bad \rightarrow \\ &\exists Ra, nl. s.runs(Ra) = (Init, A, B, Kab\#Ts\#Ta\#nl) \end{aligned} \quad (10)$$

When trying to prove that this is an invariant, we get stuck in a proof state that suggests replacing the honesty of  $A$  and  $B$  by the secrecy of  $Kab$ . This enabled the successful completion of the proof.

$$\begin{aligned} M3_{krb2} &\equiv \{s \mid \forall Kab, A, B, Ts, Ta. \\ &\text{Secure}(S, B, [Kab, A, Ts]) \in s.chan \wedge dAuth(Kab, [A, Ta]) \in s.chan \wedge \\ &Kab \notin ikk(s.chan) \rightarrow \\ &\exists Ra, nl. s.runs(Ra) = (Init, A, B, Kab\#Ts\#Ta\#nl) \} \end{aligned}$$

To establish the required guard strengthening (10) using the invariant  $M3_{krb2}$ , it suffices to show that its first premise (i.e.,  $\text{Secure}(S, B, [Kab, A, Ts]) \in s.chan$ ) and third premise (i.e., the honesty of  $A$  and  $B$ ) imply  $Kab \notin ikk(s.chan)$ . This follows from the invariant  $M2b_{krb2}^+$  above and the following invariant, which guarantees to the server that the intruder never learns a session key generated for honest agents.

$$\begin{aligned} ikkSv_{krb2} &\equiv \{s \mid \forall Rs, A, B, nl. \\ &s.runs(Rs) = (Serv, A, B, nl) \wedge A \notin bad \wedge B \notin bad \rightarrow Rs\$sk \notin ikk(s.chan) \} \end{aligned}$$

This completes our sketch of the refinement proof of  $step5_{krb1}$  by  $step5_{krb1}$ .