

Refining Key Establishment

Conference Paper**Author(s):**

Sprenger, Christoph; Basin, David

Publication date:

2012

Permanent link:

<https://doi.org/10.3929/ethz-a-007593074>

Rights / license:

In Copyright - Non-Commercial Use Permitted

Originally published in:

<https://doi.org/10.1109/CSF.2012.21>

Refining Key Establishment

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

Abstract—We use refinement to systematically develop a family of key establishment protocols using a theorem prover. Our development spans four levels of abstraction: abstract security properties, message-less guard protocols, protocols communicating over channels with security properties, and protocols secure with respect to a Dolev-Yao intruder. The protocols we develop are Needham-Schroeder Shared Key, the core of Kerberos 4 and 5, and Denning Sacco, and include realistic features such as key confirmation, replay caches, and encrypted tickets. Our development highlights that message-less guard protocols provide a fundamental abstraction for bridging the gap between security properties and message-based protocol descriptions. It also shows that the refinement approach presented in [1] can be applied, with minor adaption, to families of key establishment protocols and that it scales to protocols of nontrivial size and complexity.

I. INTRODUCTION

The fact that the development of even simple security protocols is error prone motivates the use of formal methods to ensure their security. The past decade has witnessed significant progress in post-hoc verification methods for protocol security based on model checking and theorem proving such as [2]–[5]. 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, meaning that the development is decomposed into smaller steps that are easy to understand. These steps should span well-defined abstraction levels leading the developer from the requirements down to cryptographic protocols. The resulting protocols should be secure in well-established attacker models and such claims should 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 [6]–[11] fall short of at least one of these desiderata.

We have recently proposed a method for developing security protocols by stepwise refinement so that they are correct by construction [1]. The method consists of a four-level refinement strategy, summarized in Table I, which allows the developer 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. These

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

Table I
REFINEMENT LEVELS

include (reachability-based) secrecy and authentication. We embedded this method in the theorem prover Isabelle/HOL and previously used it to derive some basic unilateral entity authentication protocols and a simplified Otway-Rees key transport protocol without key confirmation.

In this paper, we show how to systematically develop an entire family of key transport protocols. This family consists of the Needham-Schroeder Shared-key protocol (NSSK), the core of Kerberos 4 and Kerberos 5, and the Denning-Sacco protocol. Compared to the protocols developed in [1], these protocols are significantly more complex in size and message structure and exhibit a number of 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.

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 communicate with other agents by accessing their local stores. This abstraction simplifies proofs, but 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 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.

Our contributions are threefold. First, 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.¹ Our formalization includes a new infrastructure with general Isabelle/HOL theories for protocol runs, fresh values, and channels with security properties. This supersedes the protocol-specific embeddings of these concepts in [1].

Second, our development provides evidence that guard protocols constitute a fundamental abstraction that bridges the gap between security properties and standard message-based protocol descriptions. 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 security proofs. Moreover, in message-based 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 in a straightforward way. These features of guard protocols facilitate the formal development of secure protocols and underscore the central role they can play in property-driven design approaches.

Third, our development shows that our method scales to protocols of realistic complexity. We only need one extension: to model key confirmation at Level 2, we have added dynamically created channels to our modeling infrastructure. The protocols in our development share both structure and properties as the graph of refinements of our development indicates (see Figure 3). Property preservation through refinements avoids proof duplication and fosters well-structured proofs.

II. PRELIMINARIES

A. Isabelle/HOL and notation

Isabelle is a generic, tactic-based theorem prover. We have used Isabelle's implementation of higher-order logic,

¹Our entire development including the infrastructure theories is available at <http://people.inf.ethz.ch/csprengel/csf12>.

Isabelle/HOL [12], for our development. To enhance readability, we will use standard mathematical notation where possible and blur the distinction between types and sets.

We use two definitional equalities: \equiv for terms and \triangleq for types. We define partial functions by $A \rightarrow B \triangleq A \rightarrow B_\perp$, where $B_\perp \triangleq B \sqcup \perp$. The term $f(x \mapsto y)$ denotes the function that behaves like f , except it maps x to y . For a function or binary relation $R \subseteq A \times B$ and 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 formed 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 $m \in \text{multiset}(A)$, the term $m(e)$ denotes the multiplicity of e in m . Record types may be defined, e.g., $\text{point} \triangleq \langle x \in \mathbb{N}, y \in \mathbb{N} \rangle$ with elements like $r \equiv \langle x = 1, y = 2 \rangle$ 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 point with a color field. For record types T and U including fields F , we define $\Pi_F \equiv \{(r, s) \in T \times U \mid \bigwedge_{x \in F} r.x = s.x\}$. If U has exactly the fields F , the function $\pi_F: T \rightarrow U$ projects T to U .

B. Specifications and refinement

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 summarize our refinement theory that we developed in Isabelle/HOL. It is inspired by [13], [14].

Our models are *specifications* of the form $S = (T, \text{obs})$, where $T = (\Sigma, \Sigma_0, \rightarrow)$ is a *transition system* and $\text{obs}: \Sigma \rightarrow \mathcal{O}$ is an *observation function*. The state space Σ is a record type, i.e., a set of tuples of state variables, $\Sigma_0 \subseteq \Sigma$ are the initial states, and the transition relation \rightarrow is a finite union of parametrized relations, called *events*. Events have the form

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

where $\bar{\cdot}$ denotes vectors. $G(\bar{x}, s)$ 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 state s , yielding state s' .

Example 1. Consider an abstract file transfer protocol specification $S_f \equiv ((\Sigma_f, \Sigma_f, \rightarrow_f), \text{id})$, where $\Sigma_f \triangleq \langle f \in \text{file} \rangle$ with $\text{file} \triangleq I \rightarrow D$ for a finite index set I and a set of data blocks D and $\rightarrow_f \equiv xfer_f$. The event $xfer_f \equiv \{(s, s') \mid s'.f := g\}$ transfers a given file g in one shot to f . 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.

Our notion of refinement is based on standard simulation [15]. We say $S_c = (T_c, obs_c)$ refines $S_a = (T_a, obs_a)$ 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 $evt_c(\bar{x})$ (with guards G_c , state variables \bar{v} , and update functions \bar{f}_c) we identify an abstract event $evt_a(\bar{z})$ (with guards G_a , state variables \bar{u} , and update functions \bar{f}_a) simulating it, i.e., $R; evt_c(\bar{x}) \subseteq evt_a(\bar{w}(\bar{x})); R$, where ‘;’ is relational composition and $\bar{w}(\bar{x})$ are the witnesses that construct parameters for evt_a from those of evt_c . This condition decomposes into two proof obligations, called *guard strengthening* and *action refinement*, both under the premises $(s, t) \in R$ and $G_c(\bar{x}, t)$.

- $G_a(\bar{w}(\bar{x}), s)$ (GRD)
- $(s \parallel \bar{u} := \bar{f}_a(\bar{w}(\bar{x}), s), t \parallel \bar{v} := \bar{f}_c(\bar{x}, t)) \in R$ (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. Finally, (3) R respects observations mediated by π , i.e., $obs_a(s) = \pi(obs_c(t))$ for all $(s, t) \in R$. We say S_c refines S_a using π , written $S_c \sqsubseteq_{\pi} S_a$, if $S_c \sqsubseteq_{R,\pi} S_a$ for some R .

We consider two types of invariants: internal ones are supersets of $reach(S)$, the set of states reachable from initial states, and external ones are supersets of $oreach(S) \equiv obs(reach(S))$. We use internal invariants to strengthen simulation relations in refinement proofs. The following result implies that requirements and assumptions, once established as external invariants, are preserved by subsequent refinements. This does not generally hold for internal invariants.

Proposition 2. Suppose $S_2 \sqsubseteq_{\pi} S_1$ and $J \subseteq \Sigma_1$. Then we have

- 1) $oreach(S_1) \subseteq J$ implies $\pi(oreach(S_2)) \subseteq J$ and
- 2) $S_3 \sqsubseteq_{\rho} S_2$ implies $S_3 \sqsubseteq_{\rho \circ \pi} S_1$.

The observation functions allow us to relate specifications with completely different state spaces. The mediator function enables the addition of details to observations during refinement. Proposition 2 guarantees a well-defined notion of property preservation for series of refinements.

Example 3. We define a “protocol” implementing the file transfer specification S_f by $S_p \equiv ((\Sigma_p, \Sigma_{p,0}, \rightarrow_p), obs_p)$, where $\Sigma_p \triangleq \Sigma_f \parallel (b \in I \rightarrow D)$ extends the state Σ_f with a buffer b . The set $\Sigma_{p,0}$ consists of initial states of the form $(f = f_0, b = \emptyset)$ for some $f_0 \in file$ and the empty buffer. The protocol non-deterministically transfers blocks of the file g into the buffer b from where it is assigned to f , once the transfer is complete. The transition relation $\rightarrow_p \equiv xfer_p \cup \bigcup_{i \in I} blk(i)$ is the union of two events:

$$blk(i) \equiv \{(s, s') \mid i \in I \wedge s'.b := s.b(i \mapsto g(i))\}$$

and $xfer_p \equiv \{(s, s') \mid dom(b) = I \wedge s'.f := s.b\}$. The observation function $obs_p \equiv \pi_f$ projects the state Σ_p to Σ_f .

Let us try to establish a refinement between S_p and S_f , using the simulation relation $R \equiv \Pi_f$, 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 $blk(i)$ 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 $blk(i)$ and *skip* (the identity relation) requires showing $(s, t') \in \Pi_f$ for $t' = t \parallel (b := t.b(i \mapsto g(i)))$, assuming $(s, t) \in \Pi_f$ and $i \in I$. This holds trivially, since $t'.f = t.f = s.f$. In the action refinement for $xfer_p$ and $xfer_f$, we must show that $(s \parallel f := g, t \parallel f := s.b) \in \Pi_f$ assuming $(s, t) \in \Pi_f$ and $dom(b) = I$. To prove this, we need additional information about the relation between b and g , expressed as the internal invariant $I_p \equiv \{s \in \Sigma_p \mid \forall i \in dom(b). s.b(i) = g(i)\}$ 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.

III. SECURITY PROTOCOL REFINEMENT

We present a framework to support security protocol development by refinement based on the four-level refinement strategy presented in [1] (cf. Table I). In particular, we introduce infrastructure for the modeling and reasoning about protocol runs, fresh values, and channels with security properties, which replaces respective protocol-specific embeddings in [1]. Later we instantiate this framework for concrete protocol developments.

A. General setup

We define a type *agent* of agents. We assume a subset *bad* of dishonest agents, whose complement is the set of *honest* agents, an honest server $S \notin bad$, and an intruder $i \in bad$ 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 data type of freshness identifiers as $fresh \triangleq mkf(rid, \mathbb{N})$, which has a single constructor mkf . We write $mkf(R, i)$ 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.

$$nonce \triangleq fresh \quad key \triangleq sesK(fresh) \mid ltkK(ltk)$$

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. We use numbers 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 $keySetup \subseteq key \times agent$ defining the initial key knowledge of each agent. The definitions of this relation and the type ltk are deferred to Level 3 (Section III-E), since they are protocol-dependent, e.g., a 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: $corrKey \equiv keySetup^{-1}(bad)$.

B. Security properties (Level 0)

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 its 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 Proposition 2).

Secrecy : We introduce two state variables, kn and az , both relations between data (of polymorphic type δ) and agents, where $(d, A) \in s.kn$ 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.

$$\Sigma_{s0}(\delta) \triangleq \langle \langle kn \in \mathcal{P}(\delta \times agent), az \in \mathcal{P}(\delta \times agent) \rangle \rangle$$

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

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

We allow any state satisfying this property as an initial state.

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

$$\begin{aligned} gen_{s0}(d, A, G) &\equiv \{(s, s') \mid \\ &d \notin dom(s.kn) \wedge A \in G \wedge \\ &s'.kn := s.kn \cup \{(d, A)\} \wedge \\ &s'.az := s.az \cup \{(d, B) \mid B \in G \vee G \cap bad \neq \emptyset\}\} \end{aligned}$$

The guards require that d is fresh and that A belongs to the group G . The first action adds the pair (d, A) to the knowledge in $s.kn$. The second action updates the authorization relation with $\{d\} \times G$ if all agents in G are honest and with $\{d\} \times agent$ 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 agent B learns the secret d , provided that B is authorized to learn d .

$$\begin{aligned} learn_{s0}(d, B) &\equiv \{(s, s') \mid \\ &(d, B) \in s.az \wedge s'.kn := s.kn \cup \{(d, B)\}\} \end{aligned}$$

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

Proposition 4. $oreach(s0) \subseteq secrecy$.

Authentication : We formulate two models, $a0n$ and $a0i$, that represent a minimal, extensional variant of Lowe's *injective* and *non-injective agreement* using signals, which indicate particular stages of each role's progress (e.g., termination) [16]. The state record has a single field: an initially empty multiset of signals, $sigs$. The entire state is observable.

$$\begin{aligned} signal(\delta) &\triangleq \text{Running}(\text{list}(agent) \times \delta) \\ &\quad \mid \text{Commit}(\text{list}(agent) \times \delta) \end{aligned}$$

$$\Sigma_{a0}(\delta) \triangleq \langle \langle sigs \in \text{multiset}(signal(\delta)) \rangle \rangle$$

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 δ 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 .

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).

$$\begin{aligned} niagree_{a0n} &\equiv \{s \mid \forall h, d. h \cap bad = \emptyset \wedge \\ &s.sigs(\text{Commit}(h, d)) > 0 \rightarrow s.sigs(\text{Running}(h, d)) > 0\} \end{aligned}$$

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.

$$\begin{aligned} iagree_{a0i} &\equiv \{s \mid \forall h, d. h \cap bad = \emptyset \\ &\rightarrow s.sigs(\text{Commit}(h, d)) \leq s.sigs(\text{Running}(h, d))\} \end{aligned}$$

The models $a0n$ and $a0i$ each have two events, $running(h, d)$ and $commit(h, d)$, which add the corresponding signal to the multiset $s.sigs$. A guard in each $commit(h, d)$ event ensures that adding a $\text{Commit}(h, d)$ signal to $s.sigs$ preserves the respective property above. The guard in $commit_{a0n}(h, d)$ requires the existence of a running signal if the agents in h are honest. In $commit_{a0i}(h, d)$, this is strengthened as follows.

$$h \cap bad = \emptyset \rightarrow s.sigs(\text{Commit}(h, d)) < s.sigs(\text{Running}(h, d))$$

It is easy to see that $a0i$ refines $a0n$.

Proposition 5. We have that (i) $oreach(a0n) \subseteq niagree_{a0n}$, (ii) $oreach(a0i) \subseteq iagree_{a0i}$, and (iii) $a0i \sqsubseteq_{id, id} a0n$.

C. Guard protocols (Level 1)

We introduce protocol roles and runs. A run is a thread that is executed by some agent in a given role. Each run has a local memory holding state information. At this abstract level, runs communicate by reading each other's memory. We call such protocols *guard protocols*. At Level 2, we will refine this abstract but unrealistic memory-reading communication by introducing message passing over communication channels.

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.

$$\begin{aligned} \text{role} &\triangleq \text{Init} \mid \text{Resp} \mid \text{Serv} \\ \text{frame} &\triangleq [\text{atom}] \\ \text{runs}T &\triangleq \text{rid} \rightarrow \text{role} \times \text{agent} \times \text{agent} \times \text{frame} \\ \Sigma_1 &\triangleq \langle \text{runs} \in \text{runs}T \rangle \end{aligned}$$

Here, we have schematically defined the state of a Level 1 protocol by the record type Σ_1 . In concrete models, this state may contain additional variables. We assume that (at least) the variable *runs* is observable, i.e., part of the observation. All later refinements inherit the variable *runs*, but may add atoms to the run's frames.

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 $\text{runs}(R) = (\text{Init}, A, B, [Nb])$ expresses that the event executes a step of the run *R*, which is owned by agent *A* playing the initiator role, talks to the responder *B*, and has recorded a nonce *Nb* in its frame. An event's action typically extends the run frame with additional atomic messages, thereby marking the run's progress. We call a run *completed* if there is no event that extends its frame.

Agents communicate by reading their peers' memories. This is achieved by *non-local guards* that refer to another run's store. Such guards compare local and remote values and read new remote values that may be 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 III-B) executed by a run *R* requires the existence of a run *R'* executing a different role from *R* and agreeing with *R* on data *d*, provided the agents in *h* are honest. For example, the following guard in an event of run *R* requires that there is a responder run *R'* by *B* with *A* that agrees with *R* on *Na*, provided both *A* and *B* are honest.

$$B \notin \text{bad} \wedge A \notin \text{bad} \rightarrow \exists R'. \text{runs}(R') = (\text{Resp}, B, A, [Na])$$

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

We establish the secrecy and authentication properties of our guard protocol models by refining appropriately

channel type	dot notation	channel message
insecure	$A \rightarrow B : M$	$\text{Insec}(A, B, M)$
secure (static)	$A \bullet \rightarrow \bullet B : M$	$\text{Secure}(A, B, M)$
authentic (dynamic)	$A \bullet \xrightarrow{K} B : M$	$\text{dAuth}(K, M)$

Table II
CHANNEL NOTATION AND MESSAGES (SELECTION)

instantiated secrecy and authentication models from Section III-B. In each case, we establish a data 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).

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

$$\pi_{s01}(t) \equiv \langle \text{kn} = \text{kn}C(t.\text{runs}), \text{az} = \text{az}C(t.\text{runs}) \rangle$$

We can now explain how authorization guards are stated in terms of $\text{az}C$: we use the expression $(d, A) \in \text{az}C(t.\text{runs})$ to check whether an agent *A* is allowed to access data *d*.

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

$$\pi_{a01}(t) \equiv \langle \text{sigs} = \text{sigs}C(t.\text{runs}) \rangle.$$

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 $\text{sigs}C(t.\text{runs})$ contains a $\text{Commit}(h, d)$ signal for each run of *A* in role *R* where the data *d* is known and one $\text{Running}(h, d)$ 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 $\text{kn}C$, $\text{az}C$, and $\text{sigs}C$ depend on the protocol.

D. Channel protocols (Level 2)

We model protocols using communication channels with associated security properties. For informal use, we adopt the notation of [17] (Table II). 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. A *static secure* channel $A \bullet \rightarrow \bullet B$ provides *confidentiality* to *A* (only *B* can read messages) and *authenticity* to *B* (only *A* can send messages). On a *dynamic authentic* channel $A \bullet \xrightarrow{K} B : M$, *A* can authentically transmit messages to *B* provided they both know the key *K*. Such channels are created by dynamically generated (session) keys. We omit other channel types that are not needed here.

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.

$$security \triangleq \mathcal{P}(\{auth, confid\})$$

$$chmsg \triangleq \begin{array}{l} \text{StatCh}(security, agent, agent, [atom]) \\ | \text{DynCh}(security, key, [atom]) \end{array}$$

Static channel messages name the sender and the receiver. In 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 $A \bullet \xrightarrow{K} B$) only serve as an indication of the intended communication partners.

For practical use, we employ abbreviations such as those given in the third column of Table II. For example, we define

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

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

Based on the security attributes of channel messages we define the intruder's capabilities for *eavesdropping* (or *extracting*) payload messages and *faking* channel messages. We formalize these capabilities as two functions

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

where the parameter $T \subseteq atom$ specifies the intruder's initial knowledge and the parameter *U* denotes a set of run identifiers. These functions are defined by the rules in Figures 1 and 2. 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 condition $K \in rkey(U)$ in the third rule in Figure 2 restricts the intruder to using a key in

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

to fake a non-authentic dynamic message. Below, we will use $U = \text{dom}(s.runs)$ to preserve the invariant that all identifiers $R\$i$ for $R \notin \text{dom}(s.runs)$ are indeed fresh.

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

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

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

Figure 1. Rules defining extractable atoms

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

Figure 2. Rules defining fakeable channel messages

The protocol events use guards of the form $M \in s.chan$ 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 $s'.chan := s.chan \cup \{M\}$.

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

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

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

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

The refinement of the abstract Level 1 model is typically by superposition, that is, the simulation relation is based on the canonical projection $\pi_{12} : \Sigma_2 \rightarrow \Sigma_1$. The event $fake_2$ refines *skip*, since it only modifies the channel messages.

E. Cryptographic protocols (Level 3)

We model concrete protocols and the Dolev-Yao intruder using a standard theory of *cryptographic messages* due to Paulson [18]. The type of messages, *msg*, is defined inductively from agents $A \in agent$, nonces $N \in nonce$, keys $K \in key$, pairs $\langle M_1, M_2 \rangle$, and encryptions/signatures $\{M\}_K$.

For this paper, we define the set of long-term keys by $ltk \triangleq \text{shrK}(agent)$. The term $\text{shr}(A) \equiv \text{ltK}(\text{shrK}(A))$ denotes the symmetric key that *A* shares with the server *S*. We also define $keySetup \equiv \{(\text{shr}(A), C) \mid C = A \vee C = S\}$ and can therefore prove that $corrKey = \text{shr}(bad)$.

To formalize protocol properties and the intruder, we use the standard closure operators *parts*, *analz*, and *synth* on sets of messages. The term *parts*(*H*) closes *H* under sub-messages (i.e., subterms of messages), *analz*(*H*) closes *H* under submessages accessible by projection and decryption

using the keys in H , and $\text{synth}(H)$ closes H under message compositions.

Cryptographic protocols replace the channel messages chan with a variable IK (for intruder knowledge) denoting a set of cryptographic messages. Hence, like channel protocols, they extend the state Σ_1 of the refined guard protocol.

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

Initially, the set IK contains the intruder's initial knowledge, e.g., the long-term keys of all bad agents, corrKey .

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 $\text{synth}(\text{analz}(s.IK))$.

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

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

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

The relation R_{23}^{msgs} expresses that the abstractions of concrete message parts in $t.IK$ are contained in the channel variable $s.\text{chan}$. 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. 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 variables.

IV. REQUIREMENTS AND ASSUMPTIONS

Our informal requirements and assumptions for (server-based) key transport 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 VI 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.

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 (synth) and send messages composed from parts obtained by decomposing (analz) 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.

V. DEVELOPMENT OVERVIEW

We concretize our refinement strategy for deriving different server-based key establishment protocols: the Needham-Schroeder Shared-Key (NSSK) protocol [19], the Denning-Sacco protocol [20], and core versions of Kerberos 4 [21] and Kerberos 5 [22]. Figure 3 summarizes our development: each node represents a model and each arc $m \rightarrow m'$ represents a refinement $m \sqsubseteq_{\pi} m'$ for a given mediator function π (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 III-B. 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 $kt1$ provides no guarantee of the session key's authenticity, freshness, or recentness. Hence, we refine $kt1$ into further guard protocols that establish authentication properties (**R3**, **R4**) and use *freshness identifiers* [23], 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 $kt1$ into two different models that realize server authentication (**R3**). In the first model, $kt1in$, the initiator injectively agrees with the server on the

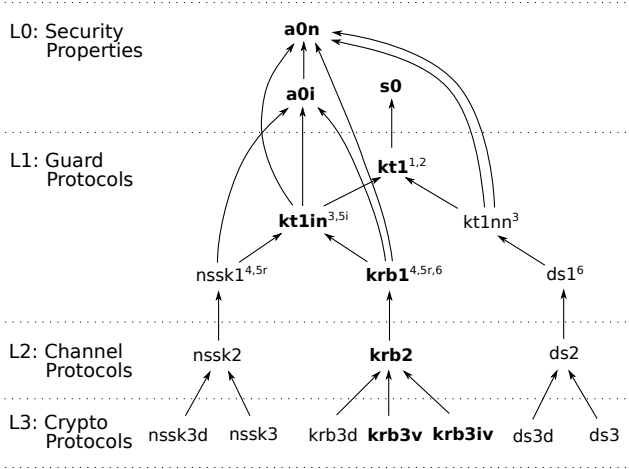


Figure 3. Refinement graph

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

Table III
L3 MODELS AND THEIR PROPERTIES

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, $kt1nn$, both initiator and responder achieve non-injective agreements with the server. The “additional data” agreed upon 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 the existence of multiple paths between some models in Figure 3.

In the second stage, we refine the model $kt1nn$ 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 the 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 on 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 $nssk3d$, $ds3d$, 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 III 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’ achieve the same properties as their listed siblings.

Based on the modeling and reasoning framework and the infrastructure from Section III, in the following sections we develop concrete models at each abstraction level. We focus on the models typeset in boldface in Figure 3 leading to the core versions of Kerberos. Figure 4 provides an overview of most refinements between these models and the related propositions. We have not included the refinements of the authentication models $a0n$ and $a0i$ by the models $kt1nn$ and $krb1$ (Propositions 7 and 10). These exhibit a structure similar to the refinement of $s0$ by $kt1$ (cf. Section III-C). We have also omitted the refinement of $krb2$ into core Kerberos 4 ($krb3iv$) stated in Proposition 13 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 IX-C, where we discuss the security guarantees that the refinements yield for the final models at Level 3.

VI. 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 III-B. We will later show that our guard protocol models (L1) refine these instantiated models, thus establishing the respective requirements (by Proposition 2). We will formalize key freshness and key recentness as invariants of Level 1 protocols and therefore discuss these later (Section VII).

Secrecy: The instantiation of the polymorphic type of data of the model $s0$ to keys provides an abstract model of

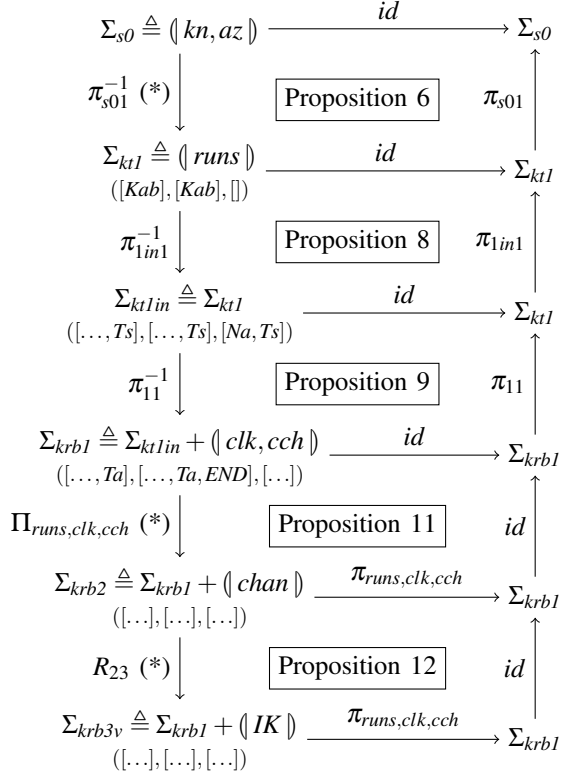


Figure 4. Details of refinements in the Kerberos development: state spaces (without variable types) and simulation relations (left), observations and mediator functions (right), and observation functions (left-to-right arrows). Triples of the form (i, r, s) describe the frames of completed initiator, responder, and server runs, where \dots stands for the fields inherited from the model above. A star (*) means that the related refinement proof requires invariants to strengthen the simulation relation.

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 to be agreed upon. We use *authentication graphs* to represent this information visually. Figure 5 displays the authentication graph of the Kerberos protocols. In these graphs, there is one node for each protocol role. Each node is labeled 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 $a0i$ 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

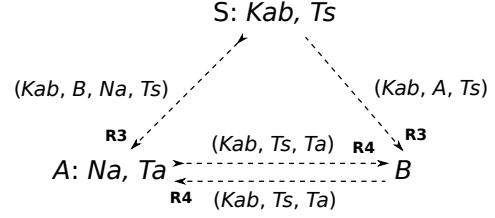


Figure 5. Authentication graph for Kerberos

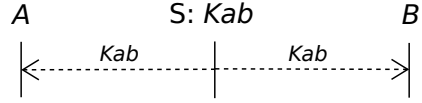


Figure 6. Basic secret key distribution ($kt1$)

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 3).

VII. 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 III-C.

$$\Sigma_{kt1} \triangleq \langle runs \in runsT \rangle$$

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

A. Secret key distribution

A sequence chart of our first abstract key transport protocol model, $kt1$, appears in Figure 6. 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 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 Kab . 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 $kt1$, we discuss the simulation relation used in the refinement

of model $s0$, which establishes session key secrecy. As described in Section III-C, 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.

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

Here, $Rs\$sk$ is the fresh value used by the server run Rs for the session key and sk is an arbitrary natural number constant. 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.

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

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

The specification $kt1$ 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 $gen_{s0}(Kab, S, [S, A, B])$.

$$\begin{aligned} step3_{kt1}(Rs, A, B, Kab) \equiv & \{(s, s') \mid \text{-- } S, \text{refines } gen_{s0} \\ & Rs \notin dom(s.runs) \wedge \text{-- fresh server run} \\ & Kab = \text{sesK}(Rs\$sk) \wedge \text{-- session key} \\ & s'.runs := s.runs(Rs \mapsto (\text{Serv}, A, B, [])) \} \end{aligned}$$

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 .

$$\begin{aligned} step5_{kt1}(Rb, A, B, Kab) \equiv & \{(s, s') \mid \text{-- } B, \text{refine } learn_{s0} \\ & s.runs(Rb) = (\text{Resp}, A, B, []) \wedge \text{-- } B's \text{ run} \\ & (Kab, B) \in azC(s.runs) \wedge \text{-- check authorization} \\ & s'.runs := s.runs(Rb \mapsto (\text{Resp}, A, B, [Kab])) \} \end{aligned}$$

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

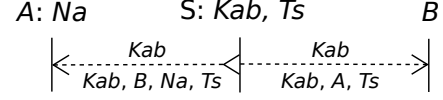


Figure 7. Adding server authentication ($ktlin$)

constraints on Kab . The authorization guard is sufficient to preserve the secrecy of Kab . In Section VII-B, 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 III-C 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 6. 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 4) is inherited by $kt1$ (Proposition 2), which thus realizes secret key distribution (**R1**, **R2**).

B. Server authentication

We now refine the model $kt1$ into $ktlin$ 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 $ktlin$. However, for the sake of the presentation, we will focus our attention on the instantiation of $ktlin$ for the Kerberos development. The models $kt1$ and $ktlin$ have identical state spaces, but in $ktlin$ 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 7 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 5. 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 $ktlin$ refines $kt1$, $a0i$, and $a0n$ (cf. Figures 3 and 4). We establish key freshness (**R5**) for the initiator as an invariant (cf. Appendix B for the similar responder case).

We obtain the model $ktlin$ by modifying $kt1$ in two ways. First, we introduce new event parameters and corresponding state updates to reflect that both partners of the 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 $R_s \mapsto (\text{Serv}, A, B, [Na, Ts])$. 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 $R_b \mapsto (\text{Resp}, A, B, [Kab, Ts])$ 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 $\text{sig}C^{rs}$, which reconstructs signal multisets from protocol runs, and use it to obtain the mediator function π_{a01}^{rs} and simulation relation R_{a01}^{rs} , as described in Section III-C. The multiset $\text{sig}C^{rs}(r) \equiv m_r$ 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.

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

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 step3_{ktlin} refines the abstract event $\text{running}_{a0n}([B, S], (Kab, A, Ts))$ and that the responder's event step5_{ktlin} refines the abstract *a0n* event $\text{commit}_{a0n}([B, S], (Kab, A, Ts))$. The remaining events refine *skip*. In the proof of guard refinement (**GRD**) for step5_{ktlin} , we get stuck in a proof state that directly suggests the following authentication guard for this event.

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

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 discover the authentication guard for $\text{step4}_{ktlin}(Ra, A, B, Na, Kab, Ts)$ in a similar manner in the proof that this event refines $\text{commit}_{a0i}([A, S], (Kab, B, Na, Ts))$.

$$A \notin \text{bad} \rightarrow \exists Rs, nl. (Kab = \text{sesK}(Rs\$sk) \\ \wedge s.\text{runs}(Rs) = (\text{Serv}, A, B, Na\#Ts\#nl)) \quad (3)$$

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

Proposition 7. *We have $ktlin \sqsubseteq_{\pi_{a01}^{rs}} a0i$ for the initiator and $ktlin \sqsubseteq_{\pi_{a01}^{rs}} a0n$ for the responder.*

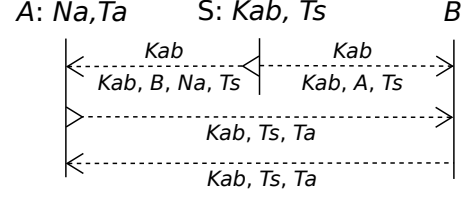


Figure 8. Adding key confirmation (*krb1*)

Finally, it is easy to see that *ktlin* refines *ktl*. The mediator function π_{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 8. $ktlin \sqsubseteq_{\pi_{1in1}^{-1}, \pi_{1in1}} ktl$.

C. Key confirmation

We next extend the model *ktlin* to an abstract model of Kerberos (Figure 8), which achieves key confirmation (**R4**), key freshness for the responder (**R5**), and key recentness (**R6**). In order 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 5). 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 8 contains all agreements specified in Figure 5 and (partially) orders them causally.

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

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

The variable clk models the discrete-time clock. We introduce an associated $\text{tick}(T)$ 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 $\text{purge}(B)$ event removes from B 's cache the entries whose timestamps Ta have expired and thus are no longer valid, which is the case when $s.\text{clk} \geq Ta + La$.

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

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.

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

The first guard generates a timestamp Ta . 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.

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

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.

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

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

Proposition 9. $krb1 \sqsubseteq_{\pi_{11}^{-1}, \pi_{11}} kt1in$.

The mediators π_{a0i}^{ir} and π_{a0i}^{ri} and associated simulation relations for refining $a0i$ and $a0n$ are defined analogously to Section VII-B. 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 a responder B knowing a key Kab and a timestamp Ta has an entry (B, Kab, Ta) in the replay cache during Ta 's validity. Appendix A provides some proof details.

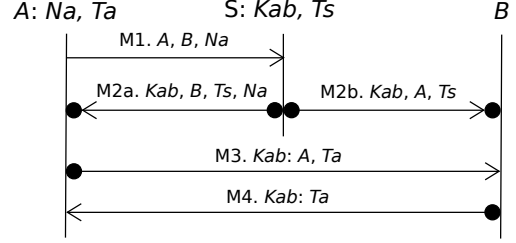


Figure 9. Channel-based Kerberos protocol ($krb2$)

Proposition 10. (i) $krb1 \sqsubseteq_{\pi_{a0i}^{ir}} a0n$ for the initiator and (ii) $krb1 \sqsubseteq_{\pi_{a0i}^{ri}} a0i$ for the responder.

Finally, in Appendix B, we formalize key freshness (**R5**) for the responder and sketch the proof that it is an invariant of $krb1$.

VIII. CHANNEL PROTOCOLS (L2)

As described in Section III-D, 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 Σ_{krb2} to Σ_{krb1} .

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

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 actions for sending channel messages. Local guards remain the same. Moreover, we introduce an active intruder as described in Section III-D. 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 9. 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.

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

In Step 5, the responder B receives message M2b from the server and M3 from the initiator A and sends message

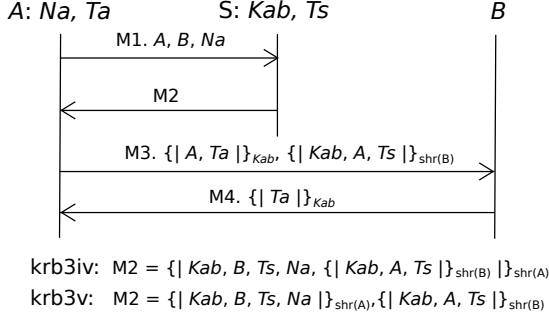


Figure 10. Cryptographic core Kerberos protocols

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 C contains a proof sketch of guard strengthening for Step 5.

Proposition 11. 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 $krb2 \sqsubseteq_{R'_{12}, id} krb1$.

IX. CRYPTOGRAPHIC PROTOCOLS (L3)

As described in Section III-E, 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 III-E 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.

Moreover, by refining the channel-based intruder into a standard Dolev-Yao intruder, as described in Section III-E, we 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 10 shows the core of Kerberos 4 [21] and Kerberos 5 [22]. 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 $krb3iv$.

A. 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 .

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

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

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

The definition of the message abstraction function, $absMsg$, is straightforward. It abstracts the components of messages $M2$ and $M3$ separately. For instance, here are the rules defining the abstraction of tickets and authenticators.

$$\begin{aligned} \frac{\{ K, A, T \}_{shr(B)} \in H}{\text{Secure}(S, B, [K, A, T]) \in absMsg(H)} \\ \frac{\{ A, T \}_K \in H}{\text{dAuth}(K, [A, T]) \in absMsg(H)} \end{aligned}$$

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 (A2). We have already established all other relevant properties on higher levels of abstraction.

Proposition 12. 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 $krb3v \sqsubseteq_{R_{23}^v, id} krb2$.

B. 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 form of message M2a, $\{K, A, T, N, X\}_{shr(A)}$, now includes a message variable X for the ticket.

The refinement proof for $krb3iv$ requires additional invariants. One of these describes the ticket's shape and the encrypted key K in message M2 sent to an honest initiator A .

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

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

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

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

Proposition 13. *Let I_{krb3iv} be the intersection of the invariants of $krb3iv$ and let $R'_{23} \equiv R'_{23} \cap (\Sigma_{krb2} \times I_{krb3iv})$. Then $reach(krb3iv) \subseteq I_{krb3iv}$ and $krb3iv \sqsubseteq_{R'_{23}, id} krb2$.*

C. 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, which states 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 4: 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 4, 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 π_{s01} (defined in Section VII-A), transforms this minimal state information to the knowledge and authorization relations *kn* and *az* of the model $s0$. Hence, using Proposition 2, the following overall secrecy result can be derived as a combination of Propositions 4, 6, 8, 9, 11, and 12.

Corollary 14. $(\pi_{s01} \circ \pi_{1in1} \circ \pi_{11})(oreach(krb3v)) \subseteq secrecy$.

In a similar way, we can express the authentication results directly as properties of $krb3v$ (cf. Figure 5). 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 15. *For the initiator, we have*

- 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 16. *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 π , Proposition 2 yields the guarantee $\pi(oreach(S')) \subseteq P$ for S' , i.e., π transforms the set of concrete observations of S' to a set of abstract observations satisfying P .

X. RELATED WORK

There have been other proposals for developing security protocols by refinement using different formalisms such as the B method [7], its combination with CSP [6], I/O automata [8], and abstract state machines (ASMs) [9]. None of these continue their refinements to the level of a full Dolev-Yao intruder. Either they only consider an intruder that is passive [8], defined ad-hoc [6], [9], or similar to our Level 2 intruder [7]. This makes a comparison of their results with standard protocol models difficult. Moreover, none provides a uniform and systematic development method such as our four-level refinement strategy and most of them develop individual protocols rather than entire families.

Mödersheim and Viganò [24] 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.

Abadi et al. [25] define a high-level process language with constructs for secure channels and compile it into a low-level language with cryptographic messages. They show a full abstraction result for their translation.

Datta et al. [11] 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. Pavlovic et al. [10], [26] similarly refine protocols by transforming messages and propose specialized formalisms for establishing secrecy and authentication properties. In

[10], they also derive core Kerberos 4 and 5 and NSSK. These refinements do not involve fundamental changes of the abstraction level since one instantiates abstract operations on messages.

Several other researchers have analyzed Kerberos. Bella and Riccobene [9] develop Kerberos 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 [27], Kerberos 4 [28], and Kerberos 5 [29] including session key compromise using the inductive approach [18]. However, they do not model a replay cache and can only prove non-injective agreements.

Butler et al. [30], [31] construct 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.

Simulation-based security [32], [33] can be seen as a paradigm for specifying idealized functionalities and refining them into protocols using a form of secure emulation. De-laune et al. [34] propose a symbolic version of this paradigm and apply it to the Needham-Schroeder-Lowe protocol.

XI. CONCLUSIONS

Our development provides strong evidence that refinement supports the systematic understanding and development of families of protocols. It also shows 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 simplifying invariant discovery. Moreover, 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).

We ultimately envision tool-based development 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 [35]. 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 5 and sequence charts of Figures 6–8. 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 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] C. Sprenger and D. Basin, “Developing security protocols by refinement,” in *Proc. 17th ACM Conference on Computer and Communications Security (CCS)*, 2010, pp. 361–374.
- [2] A. Armando et al., “The AVISPA tool for the automated validation of internet security protocols and applications,” in *Proc. Computer Aided Verification (CAV 2005)*, ser. Lecture Notes in Computer Science. Springer, 2005, vol. 3576, pp. 281–285.
- [3] D. Basin, S. Mödersheim, and L. Viganò, “OFMC: A symbolic model checker for security protocols,” *International Journal of Information Security*, vol. 4, no. 3, pp. 181–208, June 2005.
- [4] C. J. F. Cremers, “The Scyther tool: Verification, falsification, and analysis of security protocols,” in *Proc. Computer Aided Verification (CAV 2008)*, ser. Lecture Notes in Computer Science, vol. 5123. Springer, 2008, pp. 414–418.
- [5] E. Cohen, “First-order verification of cryptographic protocols,” *Journal of Computer Security*, vol. 11, no. 2, pp. 189–216, 2003.
- [6] M. J. Butler, “On the use of data refinement in the development of secure communications systems,” *Formal Aspects of Computing*, vol. 14, no. 1, pp. 2–34, 2002.
- [7] P. Bieber and N. Boulahia-Cuppens, “Formal development of authentication protocols,” in *Proc. Sixth BCS-FACS Refinement Workshop*, 1994.
- [8] N. A. Lynch, “I/O automaton models and proofs for shared-key communication systems,” in *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, 1999, pp. 14–29.
- [9] G. Bella and E. Riccobene, “Formal analysis of the Kerberos authentication system,” *Journal of Universal Computer Science*, vol. 3, no. 12, pp. 1337–1381, 1997.
- [10] I. Cervesato, C. Meadows, and D. Pavlovic, “An encapsulated authentication logic for reasoning about key distribution protocols,” in *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW)*, 2005, pp. 48–61.

- [11] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic, “A derivation system and compositionl logic for security protocols,” *Journal of Computer Security*, vol. 13, pp. 423–482, 2005.
- [12] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, ser. Lecture Notes in Computer Science. Springer, 2002, vol. 2283.
- [13] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
- [14] M. Abadi and L. Lamport, “The existence of refinement mappings,” *Theoretical Computer Science*, vol. 82, no. 2, pp. 253–284, 1991.
- [15] R. Milner, “An algebraic definition of simulation between programs,” in *Proc. 2nd international joint conference on Artificial intelligence (IJCAI)*, 1971, pp. 481–489.
- [16] G. Lowe, “A hierarchy of authentication specifications,” in *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, 1997, pp. 31–43.
- [17] 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)*, 1994, pp. 175–192.
- [18] L. Paulson, “The inductive approach to verifying cryptographic protocols,” *Journal of Computer Security*, vol. 6, pp. 85–128, 1998.
- [19] R. Needham and M. D. Schroeder, “Using encryption for authentication in large data networks of computers,” *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.
- [20] D. E. Denning and G. M. Sacco, “Timestamps in key distribution protocols,” *Communications of the ACM*, vol. 24, no. 8, pp. 533–536, 1981.
- [21] 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.
- [22] B. C. Neuman and T. Ts'o, “Kerberos: An authentication service for computer networks,” *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33–38, 1994.
- [23] L. Gong, “Variations on the themes of message freshness and replay,” in *Proc. 6th IEEE Computer Security Foundations Workshop (CSFW)*, 1993, pp. 131–136.
- [24] S. Mödersheim and L. Viganò, “Secure pseudonymous channels,” in *Proc. 14th European Symposium on Research in Computer Security (ESORICS)*, ser. Lecture Notes in Computer Science, vol. 5789. Springer, 2009, pp. 337–354.
- [25] M. Abadi, C. Fournet, and G. Gonthier, “Secure implementation of channel abstractions,” *Information and Computation*, vol. 174, no. 1, pp. 37–83, 2002.
- [26] D. Pavlovic and C. Meadows, “Deriving secrecy in key establishment protocols,” in *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*, 2006, pp. 384–403.
- [27] G. Bella and L. C. Paulson, “Mechanising BAN Kerberos by the inductive method,” in *Proc. Computer Aided Verification (CAV '98)*, ser. Lecture Notes in Computer Science, vol. 1427. Springer, 1998, pp. 416–427.
- [28] —, “Kerberos version 4: Inductive analysis of the secrecy goals,” in *Proc. 5th European Symposium on Research in Computer Security (ESORICS)*, 1998, pp. 361–375.
- [29] G. Bella, *Formal Correctness of Security Protocols*, ser. Information Security and Cryptography. Springer, 2007.
- [30] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, and C. Walstad, “Formal analysis of Kerberos 5,” *Theoretical Computer Science*, vol. 367, pp. 57–87, November 2006.
- [31] 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)*, 2002, pp. 175–.
- [32] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols,” in *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001, pp. 136–145.
- [33] M. Backes, B. Pfizmann, and M. Waidner, “The reactive simulatability (RSIM) framework for asynchronous systems,” *Information and Computation*, vol. 205, no. 12, pp. 1685–1720, 2007.
- [34] S. Delaune, S. Kremer, and O. Pereira, “Simulation based security in the applied pi calculus,” in *Proc. Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2009, pp. 169–180.
- [35] D. A. Basin and C. J. F. Cremers, “Modeling and analyzing security in the presence of compromising adversaries,” in *Proc. 15th European Symposium on Research in Computer Security (ESORICS)*, ser. Lecture Notes in Computer Science, vol. 6345. Springer, 2010, pp. 340–356.

APPENDIX

A. Step 5 in Proposition 10(ii) (L1)

We sketch the proof of guard strengthening in the refinement of the abstract event $commit_{a0i}([B, A], (Kab, Ts, Ta))$ by the concrete $step5_{krbl}(Rb, A, B, Kab, Ts, Ta)$ of the responder B , which is part of the proof that establishes the responder's injective agreement with the initiator. We define the function $sigC(r) \equiv m_r$ 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

$$c_R < c_I \quad (4)$$

assuming that A and B are honest and that the guards of $step5_{krb1}$ 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 (4) to the conjunction of the following two subgoals.

$$c_I > 0 \quad (5)$$

$$c_R = 0 \quad (6)$$

Subgoal (5) follows from the authentication guard for $step5_{krb1}$. Subgoal (6) requires that there is no responder run Rb' and list nl' corresponding to a commit signal, i.e.,

$$s.runs(Rb') = (Resp, A, B, Kab \# Ts \# Ta \# nl'). \quad (7)$$

We prove this by contradiction using an invariant stating that (7) and the validity of Ta (i.e., $s.clk < Ta + La$) entail the existence of a cache entry $(B, Kab, Ta) \in s.cch$. 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 (7) and noting that $s.clk < Ta + La$ and the replay check $(B, Kab, Ta) \notin s.cch$ are guards of Step 5, we use the invariant to derive a contradiction. This concludes the proof that the guards of the $step5_{krb1}$ event imply the guard of $commit_{a0i}$.

B. Responder's Key Freshness (L1)

Key freshness for the responder in $krb1$ expresses that a session key K appearing in a run of a responder B with an initiator A uniquely identifies that run, if A and B are honest.

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

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

Proposition 17. $oreach(krb1) \subseteq rfresh_{krb1}$.

All cases except for the critical Step 5 by the responder are proved automatically. For the case of Step 5, the proof relies on most 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 agreement with the initiator to reduce the proof to the initiator's key freshness (proved for $ktlin$).

C. Step 5 in Proposition 11 (L2)

The guard strengthening proof obligation for Step 5 requires that the guards for receiving M2b and M3 in the responder B 's $step5_{krb2}$ imply the following three non-local security guards of $step5_{krb1}$: the authorization guard

$$(Kab, B) \in azC(s.runs), \quad (8)$$

the server authentication guard

$$\begin{aligned} B \notin bad &\rightarrow \exists Rs, Na. Kab = sesK(Rs\$sk) \wedge \\ s.runs(Rs) &= (Serv, A, B, [Na, Ts]), \end{aligned} \quad (9)$$

and the initiator authentication guard

$$\begin{aligned} A \notin bad \wedge B \notin bad &\rightarrow \exists Ra, nl. \\ s.runs(Ra) &= (Init, A, B, Kab \# Ts \# Ta \# nl). \end{aligned} \quad (10)$$

The following invariant directly arises from the proof obligation expressing the strengthening of the server authentication guard (9). 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 (8).

The strengthening of the initiator authentication guard (10) 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.

$$\begin{aligned} \text{Secure}(S, B, [Kab, A, Ts]) &\in s.chan \wedge \\ dAuth(Kab, [A, Ta]) &\in s.chan \wedge \\ A \notin bad \wedge B \notin bad &\rightarrow \\ \exists Ra, nl. s.runs(Ra) &= (Init, A, B, Kab \# Ts \# Ta \# nl) \end{aligned} \quad (11)$$

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 (11) 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.

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

This completes our sketch of the refinement proof of $step5_{krb1}$ by $step5_{krb2}$.