Developing Security Protocols by Refinement

Christoph Sprenger
Dept. of Computer Science
ETH Zurich, Switzerland
sprenger@inf.ethz.ch

David Basin
Dept. of Computer Science
ETH Zurich, Switzerland
basin@inf.ethz.ch

ABSTRACT

We propose a development method for security protocols based on stepwise refinement. Our refinement strategy guides the transformation of abstract security goals into protocols that are secure when operating over an insecure channel controlled by a Dolev-Yao-style intruder. The refinement steps successively introduce local states, an intruder, communication channels with security properties, and cryptographic operations realizing these channels. The abstractions used provide insights on how the protocols work and foster the development of families of protocols sharing a common structure and properties. In contrast to post-hoc verification methods, protocols are developed together with their correctness proofs. We have implemented our method in Isabelle/HOL and used it to develop different entity authentication and key transport protocols.

Categories and Subject Descriptors

C 2.2 [**Computer-communication networks**]: Network protocols – *Protocol verification*; D 2.4 [**Software engineering**]: Software/Program verification - *Formal methods, correctness proofs.*

General Terms

Security, Design, Verification.

Keywords

Security protocols, stepwise refinement, formal development, entity authentication, key establishment.

1. INTRODUCTION

Designing security protocols is a non-trivial task and therefore an attractive target for formal methods. The vast majority of existing approaches, whether automated or interactive, symbolic or computational, are designed for post-hoc verification of completed protocol designs. This means that all decisions regarding the number of roles, the communication and message structure, and cryptographic primitives must be made before verification begins.

In this paper, we advocate a development method based on stepwise refinement, where we start from an abstract model that we gradually concretize by incorporating additional design elements (superposition) and by transforming existing ones (data refinement). Organizing developments this way has numerous benefits compared to post-hoc verification approaches. First and foremost, refinement enables us to reason abstractly about the problem and extract its essential features. This abstract analysis leads to valuable insights into the problem. Second, refinement helps us to master the complexity of both models and proofs by focusing on individual design aspects at each step. Third, refinement naturally supports the hierarchical development of protocol families through branching points in the refinement design space. For example, we may prove the secrecy of a session key for key transport protocols in an abstract model that we later refine into more concrete protocols with different communication topologies, message structures, and cryptographic primitives, without repeating the secrecy proof.

We specify each model by a transition system together with a set of invariants. The initial models constitute an abstract specification of the security properties required of our protocol. Each subsequent model transforms or extends its predecessor and may introduce additional security properties. The correctness of each refinement step is justified by a simulation proof, in the spirit of refinement in Event-B [4]. The final model is a full-fledged cryptographic protocol that is secure against a standard Dolev-Yao intruder [17]. *We have formalized and machine-checked all results presented in this paper*, namely, our theory of refinement and all protocol developments, using the Isabelle/HOL theorem prover [30].

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

Level 0 *Abstract, protocol-independent specifications of secrecy and authentication properties.* The model's state contains just enough structure to formulate these properties as invariants and realize them abstractly by just a few events. There are no intruder actions.

Level 1 *Abstract protocol without message passing.* We introduce protocol roles, local states of agents, and basic protocol steps. Agents read data directly from other agents' local state. There are still no intruder actions.

Level 2 *Protocol communicating over abstract channels with security properties*, such as confidential and authentic

CCS'10, October 4–8, 2010, Chicago, Illinois, USA.

channels. The intruder may eavesdrop messages on non-confidential channels and fake messages on non-authentic channels. No cryptography is used.

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

In order to validate the effectiveness of our refinement strategy, we developed different authentication and key establishment protocols from abstract specifications. We derived the ISO/IEC 9798-3 protocol and the first two steps of the Needham-Schroeder-Lowe protocol, which are unilateral entity authentication protocols based on signatures and public-key encryption. Here, our initial model expresses authentication in terms of a minimal structure, namely, the view of each role at the end of a protocol run. We also developed various server-based key transport protocols using symmetric-key cryptography, including a variant of the Otway-Rees protocol, the Needham-Schroeder Shared-Key protocol, the Yahalom protocol, and the core of the Kerberos IV and V protocols. For each of these protocols, we refine two initial models to establish both secrecy and authentication properties: the first model is an abstract representation of secrecy that we instantiate with session keys and the second is the entity authentication model mentioned earlier. Interestingly, the final models in each class of protocols refine a common Level 1 ancestor, even though they use different cryptographic primitives and communication patterns.

We see four main contributions in our work. The first contribution is methodological. Our initial models specify the security goals of the abstract protocol models at Level 1. These in turn determine the basic structure of entire families of protocols. Using the refinement strategy outlined above, we *systematically* refine these abstract models in a series of well-defined steps to protocols using cryptographic messages sent over an insecure channel. We illustrate this with known protocols, but we have also used this strategy to develop new variants of these protocols. Our refinement strategy aims at proving security properties at the highest possible abstraction level. General results guarantee that these properties are preserved by subsequent refinements.

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

Our second contribution is to show how to systematically model and use channels with security properties to construct and reason about security protocols. This fundamental abstraction allows us to reason about a protocol's security properties at a lower level of complexity than with the Dolev-Yao intruder. It also enables a range of different realizations. For example, we may implement an authentic channel using signatures or MACs. Moreover, the communication structure may change from Level 2 to 3. For instance, an abstract key server may independently distribute keys on secure channels to the initiator and responder, whereas in the concrete realization the distribution is sequential: one role receives two encrypted copies of the key and forwards one copy to the other role (Section 5). The abstract view represents the essential structure of server-based key distribution. The forwarding is just an implementation decision.

Our third contribution is to show how refinement can be used to develop protocols that are secure under the standard Dolev-Yao intruder model (at Level 3). In contrast, in related work such as [6, 9, 12, 23], the authors do not continue the refinements down to the level of a standard Dolev-Yao model based on an algebra of cryptographic messages; some use ad-hoc, protocol-dependent intruder definitions. This makes it difficult to compare their models with existing work on protocol modeling and verification and to know for which adversaries their properties hold.

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

2. PRELIMINARIES

2.1 Isabelle/HOL

Isabelle is a generic, tactic-based theorem prover. We have used Isabelle's implementation of higher-order logic, Isabelle/HOL [30], for our developments. To enhance readability, we will use standard mathematical notation in our presentation where possible.

We write $t\!:\!T$ for a term t of type T. We use $\equiv$ for definitional equality. Lowercase Greek letters denote type variables. The type of natural numbers is denoted by nat. Given types T and U, $T \to U$ is the type of total functions, $T \times U$ is the product type, $T\ list$ is the type of lists with elements in T, and $T\ set$ is the type of sets over T. The set of all elements of type T is denoted by $\mathcal{U}_T$. We drop the subscript T when it is clear from the context. The term $f[A]$ denotes the image of a set A under a function or binary relation f. The domain of a function or a binary relation f is denoted by $dom(f)$ and its range by $ran(f)$. The term $R; S$ denotes the forward composition of the binary relations R and S.

The keyword **datatype** is used to define an inductive data type. For example, the polymorphic option type is defined as **datatype** $\alpha_\bot = \bot \mid \mathsf{Some}(\alpha)$. To improve readability, we may omit the constructor Some. Functions of type $T \to U_\bot$ model partial functions from T to U. The term $f(x \mapsto y)$ denotes the function that behaves like f, except that it maps x to y. The declaration **types** $T = U$ defines T as an alias for U, for example, we define **types** $\alpha \rightharpoonup \beta = \alpha \to \beta_\bot$. We define multisets over α by **types** $\alpha\ multiset = \alpha \to nat$. For $m_1, m_2 : \alpha\, multiset$, the term $m_i(e)$ indicates the multiplicity of e in m_i and the term $m_1 \uplus m_2$ denotes multiset union and is defined by $(m_1 \uplus m_2)(e) = m_1(e) + m_2(e)$.

Record types may be defined, such as **record** $point = x\!: nat\ y\!:\!nat$ with elements like $r = (\!|\, x = 1, y = 2 \,|\!)$ and projections $r.x$ and $r.y$. The term $r(\!|\, x := 3 \,|\!)$ denotes r, where x is updated to 3, i.e., $(\!|\, x = 3, y = 2 \,|\!)$. To extend $point$ with a $color$ field, we define **record** $cpoint = point + c\!:\!color$.

channel type	dot notation	channel message
insecure	$A \rightarrow B : M$	$\mathsf{Insec}(M)$
confidential	$A \rightarrow\!\!\bullet\, B : M$	$\mathsf{Confid}(B, M)$
authentic	$A \bullet\!\!\rightarrow B : M$	$\mathsf{Auth}(A, B, M)$
secure	$A \bullet\!\!\rightarrow\!\!\bullet\, B : M$	$\mathsf{Secure}(A, B, M)$

Table 1: Channel notation and messages

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 $\pi_F \colon T \to U$ projects T to U.

2.2 Channels and messages

At Level 2 of our refinement strategy, we model protocols using communication channels with associated security properties. These channels transmit plaintext messages without cryptographic operations built from agent names, nonces, and keys. We define a type of agents, *agent*, containing a distinguished server S and an intruder i. We stipulate the existence of a set of dishonest agents, *bad* (with complement *good*), such that $\mathsf{i} \in \mathit{bad}$ and $\mathsf{S} \in \mathit{good}$. The types *nonce* and *key* are aliases for *nat*.

For informal use, we adopt the notation of [25] (second column of Table 1). We write $A \to B$ for an insecure channel from agent A to agent B. The ": M" indicates that the message M is sent on the channel. Security properties are indicated by a dot on one or both sides of the arrow. The respective agent has exclusive access to the marked end. A *confidential* channel $A \rightarrow\!\!\bullet\, B$ provides a service to A, namely, A knows that only B can receive messages. An *authentic* channel $A \bullet\!\!\rightarrow B$ provides a service to B, whereby B knows that only A can send messages. A *secure* channel $A \bullet\!\!\rightarrow\!\!\bullet\, B$ provides both agents with guarantees.

We formalize the *channel messages* that can be sent by a data type *chmsg*, providing one constructor for each channel type (third column of Table 1). For example, $\mathsf{Confid}(B, M)$ denotes a confidential message for B. We also say that M is sent to B on a confidential channel. The last argument of each constructor takes a non-cryptographic payload message. The constructors also contain the agent names that are relevant for the respective security property.

We formally define the intruder's capabilities in Sections 4 and 5. Informally, the intruder can *eavesdrop* messages on all insecure and authentic channels and on confidential and secure channels with a dishonest receiver. The set of these messages defines the intruder knowledge. Using this knowledge, the intruder can *fake* messages on all insecure and confidential channels as well as on authentic and secure channels with a dishonest sender *or receiver.* The asymmetry between confidential and authentic messages (the latter contain both the sender and receiver's name) is needed to allow the intruder to fake messages on a channel $A \bullet\!\!\rightarrow B$ to a dishonest B. This enables the realization of authentic messages $\mathsf{Auth}(A, B, M)$ by symmetric cryptography, where the intruder learns the symmetric key K_{AB} from B and can, for instance, produce a MAC.

At Level 3, we model concrete protocols and the Dolev-Yao intruder using a standard theory of *cryptographic messages* due to Paulson [31], which we summarize here. The type of messages, *msg*, is defined inductively. To improve readability, we replace the constructors by notational conventions: agents A, B, C, nonces N, keys K, pairs $\{\!|M_1, M_2|\!\}$, and encryptions $\{\!|\, M \,|\!\}_K$. Signatures are modeled as encryptions with private keys. The terms $\mathit{pub}(A)$, $\mathit{pri}(A)$, and $\mathit{shr}(A)$ denote A's public key, A's private key, and the symmetric key that A shares with S. To formalize protocol properties and a Dolev-Yao intruder, we use the standard closure operators *parts*, *analz*, and *synth* on sets of messages. The term $\mathit{parts}(H)$ closes H under submessages, $\mathit{analz}(H)$ closes H under submessages accessible using the keys in H, and $\mathit{synth}(H)$ closes H under message composition operators.

3. REFINEMENT IN ISABELLE/HOL

We summarize here our theory of refinement that we developed in Isabelle/HOL. It borrows elements from existing formalisms such as Event-B [4] and that of [2].

3.1 Specifications and implementation

Our models are specifications consisting of transition systems equipped with an observation function.

Definition 1. A *transition system* $T = (\Sigma, \Sigma_0, \rightarrow)$ is a triple, where Σ is the state space, $\Sigma_0 \subseteq \Sigma$ the set of initial states, and $\rightarrow \subseteq \Sigma \times \Sigma$ the transition relation. A *specification* $S = (T, O, \mathit{obs})$ extends a transition system T with a function $\mathit{obs} : \Sigma \to O$ mapping states to observations in O.

We write $s \to t$ for $(s, t) \in \rightarrow$. The set of finite behaviors of S, $\mathit{beh}(S)$, consists of the sequences of states starting in an initial state in Σ_0 and linked by transitions in $\rightarrow$. The term $\mathit{reach}(S)$ denotes the set of T's reachable states. Since we only consider safety properties (in fact, invariants), it suffices to consider just finite behaviors. The observation function *obs* specifies which state information is visible to an outside observer. The set of *observable behaviors* and *reachable observations* of the specification S are defined by $\mathit{obeh}(S) \equiv \mathit{obs}[\mathit{beh}(S)]$ and $\mathit{oreach}(S) \equiv \mathit{obs}[\mathit{reach}(S)]$, where *obs* is applied to behaviors elementwise.

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

Definition 2. We say S_2 *implements* S_1 *via the mediator function* $\pi : O_2 \to O_1$ if $\pi[\mathit{obeh}(S_2)] \subseteq \mathit{obeh}(S_1)$.

The binary relation obtained by existentially quantifying the mediator function in Definition 2 is reflexive and transitive. The mediator function π allows us to extend observations during refinement. For example, we may implement a model by adding a new variable that is observable.

3.2 Invariants and refinement

We consider two types of invariants, internal and external. A set $I \subseteq \Sigma$ is an *internal invariant* of a specification $S = (T, O, \mathit{obs})$, if $\mathit{reach}(S) \subseteq I$. Internal invariants are often needed to strengthen the simulation relation in refinement proofs. A set $J \subseteq O$ is an *external invariant* of S if $\mathit{oreach}(S) \subseteq J$. The importance of external invariants is that they are preserved by implementations.

PROPOSITION 1. *Suppose* S_2 *implements* S_1 *via* π *and let* $J \subseteq O_1$ *such that* $\mathit{oreach}(S_1) \subseteq J$*. Then* $\pi[\mathit{oreach}(S_2)] \subseteq J$*.*

There is no similar result for internal invariants in general. We developed several proof rules to establish internal and external invariants. We usually derive external invariants from internal ones that are observable in the following sense. We call a set $P \subseteq \Sigma$ *observable* if there is a set $Q \subseteq O$ such that $P = obs^{-1}[Q]$. This means that P cannot distinguish between any two states resulting in the same observation. Sometimes we also do the inverse: internalize external (for example, preserved) invariants.

PROPOSITION 2. *Let* $S = (T, O, obs)$ *be a specification and let* $P \subseteq \Sigma$ *and* $Q \subseteq O$ *such that* $P = obs^{-1}[Q]$. *Then we have* $reach(S) \subseteq P$ *if and only if* $oreach(S) \subseteq Q$.

Our notion of refinement between specifications is the standard notion of simulation, which we first define on transition systems and then extend to specifications.

Definition 3. Let T_1 and T_2 be transition systems. We say that T_2 *refines* T_1 *using the simulation relation* $R \subseteq \Sigma_1 \times \Sigma_2$, written $T_2 \sqsubseteq_R T_1$ if (1) $\Sigma_{0,2} \subseteq R[\Sigma_{0,1}]$ and (2) $R; \rightarrow_2 \subseteq \rightarrow_1; R$.

Condition (1) states that for each concrete initial state $t \in \Sigma_{0,2}$ there is an abstract state $s \in \Sigma_{0,1}$ such that $(s, t) \in R$. Condition (2) requires that each concrete transition can be simulated by an abstract one. Namely, for each pair of related states $(s, t) \in R$ and each concrete transition $t \rightarrow_2 t'$, there is an abstract state s' such that $s \rightarrow_1 s'$ and $(s', t') \in R$. In most of our refinement proofs, we use a proof rule derived from this definition that supports strengthening the simulation relation with invariants.

We extend refinement to specifications by adding a mediator function and requiring that the simulation relation is consistent with observations, as depicted below.

$$\begin{tikzcd}
O_1 & \Sigma_1 \arrow[l, "obs_1"'] \arrow[d, "R"] \\
O_2 \arrow[u, "\pi"'] & \Sigma_2 \arrow[l, "obs_2"']
\end{tikzcd}$$

Definition 4. S_2 refines S_1 using the simulation relation $R \subseteq \Sigma_1 \times \Sigma_2$ and the mediator function $\pi : O_2 \rightarrow O_1$, *written* $S_2 \sqsubseteq_{R,\pi} S_1$, *if* $T_2 \sqsubseteq_R T_1$ *and* R respects observations mediated by π, *i.e.,* $obs_1(s) = \pi(obs_2(t))$ *for all* $(s, t) \in R$. S_2 refines S_1 *via* π, *written* $S_2 \sqsubseteq_\pi S_1$, *if there is an* R *such that* $S_2 \sqsubseteq_{R,\pi} S_1$.

Refinement is reflexive (with the identity mediator function) and transitive (composing mediator functions). Moreover, refinement is a sound method to establish implementation.

PROPOSITION 3. *If* $S_2 \sqsubseteq_\pi S_1$ *then* S_2 *implements* S_1 *via the mediator function* π.

3.3 Refinement method and practice

We will work with structured states and transition relations, analogous to Event-B. We use state records, which are tuples of state variables, and specify the transition relation as a finite union of parametrized relations, which we call *events*. Here is a prototypical event with a parameter list $\overline{x}$.

$$event(\overline{x}) \equiv \{(s, s') \mid G(\overline{x}, s) \ \wedge \ s'.\overline{v} := \overline{f}(\overline{x}, s)\ \}$$

Events consist of two conjoined parts: a conjunction of *guards* $G(\overline{x}, s)$ and an *action* $s'.\overline{v} := \overline{f}(\overline{x}, s)$ with *update functions* $\overline{f}$. The guards depend on the parameters $\overline{x}$ and the current state s and determine when the event is enabled. The action denotes the relation $s' = s\llparenthesis\, \overline{v} := \overline{f}(\overline{x}, s)\, \rrparenthesis$, that is, the simultaneous assignment of values $\overline{f}(\overline{x}, s)$ to the variables $\overline{v}$ in state s, yielding state s'. Actions may generally be arbitrary relations, but assignments suffice here.

Exploiting this structure, we can prove invariant preservation separately for each event. For refinement, we can similarly show the second condition of Definition 3 eventwise. For each concrete event $evt_c(\overline{x})$ (with guards G_c, state variables $\overline{v}$, and update functions $\overline{f_c}$) we identify an abstract event $evt_a(\overline{z})$ (with guards G_a, state variables $\overline{u}$, and update functions $\overline{f_a}$) simulating it. For these two events we prove

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

where the *witnesses* $\overline{w}(\overline{x})$ construct parameters for evt_a from those of evt_c. Unfolding these events yields the following two proof obligations, called *guard strengthening* and *action refinement*, both under the premises $(s, t) \in R$ and $G_c(\overline{x}, t)$.

1. $G_a(\overline{w}(\overline{x}), s)$ **(GRD)**
2. $\big(s\llparenthesis\, \overline{u} := \overline{f_a}(\overline{w}(\overline{x}), s)\, \rrparenthesis, t\llparenthesis\, \overline{v} := \overline{f_c}(\overline{x}, t)\, \rrparenthesis\big) \in R$ **(ACT)**

We assume that the special event *skip*, namely, the identity relation, is present in all specifications. Setting evt_a to *skip* achieves a *temporal refinement*: the new event evt_c has no corresponding effect on the abstract state.

A development starts from a set of system requirements and assumptions about the environment. We then incrementally construct a system that fulfills the requirements provided it runs in an environment satisfying the assumptions. The crucial point is that requirements and assumptions, once modeled, are *preserved* by subsequent refinement steps. Requirements (or assumptions) may be established by variables, events, or external invariants. A variable is preserved if it is observable: this enables the reconstruction of abstract observations from concrete states. For events, preservation means that some observable effect of an event is preserved. External invariants are preserved by virtue of Propositions 1 and 3. The definition of observations plays a crucial role in formalization and preservation of requirements: Increasing the observable abstract state information strengthens the properties that are provably preserved by the refinement.

As with other deductive approaches, the inability to prove a property, namely an invariant or a refinement, can have three possible causes. First, the property may be provable and the human verifier is unable to find a proof. This requires a better understanding of the problem, which may lead to the formulation of simplified properties. Second, the property may not be inductive and requires strengthening to become provable. Finally, the property may fail to hold, in which case the failed proof state gives information about a possible attack. In contrast to post-hoc verification, such failures may occur at different levels of abstraction. In particular, a failed refinement proof may require adding behaviors to an abstract model that is too restrictive or removing behaviors from a concrete model.

4. AUTHENTICATION PROTOCOLS

We have used our methodology to develop two unilateral authentication protocols. Both protocols are based on a standard challenge-response pattern, where the initiator sends a nonce as a challenge and the responder returns it in a cryptographically transformed form that authenticates him to the initiator. The first protocol is the signature-based two-pass ISO/IEC 9798-3 standard [18]. The second, which we call NSL/2, consists of the first two steps of the Needham-Schroeder-Lowe protocol [21] and uses public-key encryption. Due to space limitations, we only discuss the ISO protocol here.

4.1 Requirements and assumptions

We start by specifying the system requirements and by making our assumptions about the environment explicit.

Requirement 1 (Protocol roles). The protocol has two roles, which we call initiator and responder.

Our notion of authentication is based on agreement [22]. Informally, we say that a role R *non-injectively agrees* with the role R' on the data d if whenever an honest agent A in role R terminates a run, apparently with an honest agent B in role R', then there is a run of agent B in role R' with whom he agrees on the participants, their roles, and the data d. Such an agreement is *injective* if there is at most one run of role R that agrees with a given run in role R'.

Requirement 2 (Entity authentication). The initiator injectively agrees with the responder on the initiator's nonce and possibly on additional data.

We will assume a standard Dolev-Yao intruder that we identify as usual with the communication network.

Assumption 1 (Dolev-Yao intruder). The intruder controls the network. He receives all messages sent and he can build ($synth$) and send messages composed from message parts obtained by decomposing ($analz$) received messages using the cryptographic keys he knows.

The following assumptions concern the corruption of agents by the intruder and the cryptographic setup.

Assumption 2 (Static corruption). An arbitrary fixed subset of agents is corrupted, whereby their long-term keys are exposed to the intruder.

Assumption 3 (Key distribution). The requisite cryptographic keys are distributed prior to protocol execution.

4.2 Entity authentication (L0)

Our development starts with an abstract, protocol-independent initial model, where the state contains just enough information to specify entity authentication. In particular, we follow Lowe [22] and formulate agreement in terms of signals that indicate particular stages of each role's progress, such as termination. More precisely, we present two models, one for non-injective agreement, $a0n$, and one for injective agreement, $a0i$. We show that $a0i$ refines $a0n$.

The state record has a single field $sigs$, which is a multiset of signals. In the initial state, $sigs$ is the empty multiset. The entire state is observable.

$$\begin{aligned}&\textbf{datatype}\ \delta\ signal = \mathsf{Running}(agent\ list \times \delta)\\ &\qquad\qquad\qquad\quad \mid \mathsf{Commit}(agent\ list \times \delta)\\ &\textbf{record}\ \delta\ a0\_state = sigs : \ (\delta\ signal)\ multiset\end{aligned}$$

There are two signals: $\mathsf{Running}(h, d)$ and $\mathsf{Commit}(h, d)$. Here, h is a list of agents and d is some data of polymorphic type δ, which is later instantiated depending on the protocol. The agreement on the data d will hold under the condition that the agents in h are honest. The position of an agent in the list h indicates its role. For the role R to agree with the role R', we follow the convention that the agents executing these roles appear in the first two positions of h. For example, later in this section, we will use $h = [A, B]$, where A is the initiator and B the responder. In the simplest case, h includes all participating agents. However, when it is not necessary to assume the honesty of some role, we move the corresponding agent to d (cf. Section 5.3).

Non-injective agreement [22] can be expressed as follows: if the agents in h are honest and there is a $\mathsf{Commit}(h, d)$ signal, then there is a matching running signal $\mathsf{Running}(h, d)$.

$$\begin{aligned}&a0n\_inv1\_niagree \equiv \{s \mid \forall h, d.\ h \subseteq good\ \wedge\\ &\quad s.sigs(\mathsf{Commit}(h, d)) > 0 \rightarrow s.sigs(\mathsf{Running}(h, d)) > 0\ \}\end{aligned}$$

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

$$\begin{aligned}&a0i\_inv1\_iagree \equiv \{s \mid \forall h, d.\ h \subseteq good\\ &\quad \rightarrow s.sigs(\mathsf{Commit}(h, d)) \leq s.sigs(\mathsf{Running}(h, d))\ \}\end{aligned}$$

The initial model $a0n$ has only two events. The first event adds a $\mathsf{Running}(h, d)$ signal to the multiset $s.sigs$.

$$\begin{aligned}&a0n\_running(h, d) \equiv \{(s, s') \mid\\ &\quad s'.sigs := s.sigs \uplus \{\mathsf{Running}(h, d)\}\ \}\end{aligned}$$

The second event adds a $\mathsf{Commit}(h, d)$ signal to the multiset $s.sigs$. Its guard requires that there exists a matching $\mathsf{Running}(h, d)$ signal if the agents in h are honest. This ensures that invariant $a0n\_inv1\_niagree$ is preserved. The honesty conditions weaken the guard just enough to accommodate the protocol's interaction with an explicit intruder in later refinements.

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

The model $a0i$ for injective agreement is the same except for the guard in $a0i\_commit(h, d)$, which we strengthen as follows to preserve the invariant $a0i\_inv1\_iagree$.

$$\begin{aligned}&h \subseteq good\\ &\rightarrow s.sigs(\mathsf{Commit}(h, d)) < s.sigs(\mathsf{Running}(h, d)\end{aligned}$$

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

Proposition 4. *We have* $reach(a0n) \subseteq a0n\_inv1\_niagree$, $reach(a0i) \subseteq a0i\_inv1\_niagree$, *and* $a0i \sqsubseteq_{Id,id} a0n$.

Since the variable $sigs$ is observable, these invariants will be preserved by further refinements (Propositions 1 and 3).

4.3 Abstract authentication protocol (L1)

Following our refinement strategy, we now introduce protocol runs and roles. A run is a thread, which is executed by some agent in a given role. At this level, runs communicate by reading each other's memory. We will later refine this by introducing communication channels. Moreover, there are no intruder events.

The state record $a1\_state$ contains a single field $runs$ for protocol runs, which replaces the initial models' signals.

$$\begin{array}{l}
\textbf{datatype}\ \mathit{frame} = \mathsf{Init}(\mathit{nonce}_{\perp}) \mid \mathsf{Resp}(\mathit{nonce}) \\
\textbf{record}\ \mathit{a1\_state} = \\
\quad \mathit{runs} : \mathit{rid} \rightharpoonup \mathit{agent} \times \mathit{agent} \times \mathit{frame}
\end{array}$$

The variable *runs* maps run identifiers to triples consisting of the initiator agent, the responder agent, and a run frame. Since run identifiers are freshly generated values, they can also play the role of nonces or keys. In this model, we use them as nonces.

The frame is a sum type with one constructor for each role. The (protocol-dependent) constructor arguments correspond to the different messages (such as nonces and keys) that are collected during the role's execution. We use the option type $(\cdot)_{\perp}$ for initially unavailable elements. The frame above models two roles, initiator and responder, establishing **Requirement 1**. In the protocols we consider, the initiator receives a responder nonce along with its own nonce. Thus, each role records the other's nonce in its frame. In the initial state, the *runs* field is empty. The entire state of this model is observable.

The three events of this specification abstractly model a protocol that follows a standard challenge-response pattern. The first event refines *skip* and just creates an initiator run.

$$\begin{array}{ll}
\mathit{a1\_step1}(A, B, Na) \equiv \{(s, s') \mid & \texttt{-- by } A \\
\quad Na \notin \mathit{dom}(s.\mathit{runs}) \land & \texttt{-- } Na\ \texttt{fresh} \\
\quad s'.\mathit{runs} := s.\mathit{runs}(Na \mapsto (A, B, \mathsf{Init}(\perp)))\ \} &
\end{array}$$

The second event refines *a0i_running* and creates a responder run identified by *Nb*. The run takes an *arbitrary* nonce *Na*, which need not come from an initiator. This reflects that the intruder can fake the challenge nonce in later refinements.

$$\begin{array}{ll}
\mathit{a1\_step2}(A, B, Na, Nb) \equiv \{(s, s') \mid & \texttt{-- by } B \\
\quad Nb \notin \mathit{dom}(s.\mathit{runs}) \land & \\
\quad s'.\mathit{runs} := s.\mathit{runs}(Nb \mapsto (A, B, \mathsf{Resp}(Na)))\ \} &
\end{array}$$

The third step refines *a0i_commit* and models the initiator run *Na* receiving back its nonce from a responder run *Nb*. This is described by the second guard, which we call an *authentication guard*, since it ensures an agreement, in this case with the responder on the pair of nonces (Na, Nb).

$$\begin{array}{ll}
\mathit{a1\_step3}(A, B, Na, Nb) \equiv \{(s, s') \mid & \texttt{-- by } A \\
\quad s.\mathit{runs}(Na) = (A, B, \mathsf{Init}(\perp)) \land & \\
\quad (A \notin \mathit{bad} \land B \notin \mathit{bad} \rightarrow s.\mathit{runs}(Nb) = (A, B, \mathsf{Resp}(Na))) \land & \\
\quad s'.\mathit{runs} := s.\mathit{runs}(Na \mapsto (A, B, \mathsf{Init}(Nb)))\ \} &
\end{array}$$

More precisely, if A or B is dishonest then *Nb* is arbitrary. Otherwise, *Nb* identifies a responder run that has previously received *Na*. This can be seen as reading *Nb* from the responder's store. We will eliminate this abstraction (which is not realizable in a distributed setting) in the next refinement when we introduce communication channels.

More generally, for A in role R to agree with B in role R' on data d assuming honest agents $h = [A, B, \ldots]$, we add an *authentication guard* G to an appropriate event of agent A (for example, the final event of its role). This guard requires that the agent B in role R' knows (at least) the data d, whenever the agents in h are honest. It has the form

$$G = h \subseteq \mathit{good} \rightarrow \exists \overline{x}.\ s.\mathit{runs}(\mathit{rid}) = (\overline{a}, \mathsf{R}'(\overline{t}))\,,$$

where R' is the frame constructor for (B's) role R', $\overline{t}$ is a tuple of terms, $\overline{a}$ is a tuple of agents including A and B. The free variables of G are exactly the variables appearing in h and d. All other variables are bound by the existential quantifier. For example, in *a1_step3*, we have $h = [A, B]$ and $d = (Na, Nb)$.

Let *a1* be the above model. The simulation relation describes a data refinement of the initial model's signals to the runs of this model. We map completed initiator and responder runs to Commit and $\mathsf{Running}$ signals to express agreement between these two roles on the nonces *Na* and *Nb*. This abstraction is described by the function $r2s$, which translates a run map r (such as *runs* in *a1_state*) to a multiset of signals (such as *sigs* in *a0_state*).

$$r2s(r)(S) = \begin{cases} 1 & \text{if } S = \mathsf{Commit}([A,B],(Na,Nb)) \\ & \text{and } r(Na) = (A, B, \mathsf{Init}(Nb)) \\ 1 & \text{if } S = \mathsf{Running}([A,B],(Na,Nb)) \\ & \text{and } r(Nb) = (A, B, \mathsf{Resp}(Na)) \\ 0 & \text{otherwise} \end{cases}$$

In general, this abstraction function maps each signal to the number of runs of the appropriate form associated with that signal. Since r is a partial function, there is at most one such run, whenever the run identifier appears in the signal's data, as above. As another example, suppose we would require agreement on *Na* only. The function $r2s(r)$ would then map the signal $\mathsf{Running}([A,B], Na)$ to the cardinality of the set $\{N \mid r(N) = (A, B, \mathsf{Resp}(Na))\}$. For each agreement property of the protocol, we prove a refinement of *a0i* or *a0n* by defining an appropriate abstraction function along these lines (see also Section 5.3).

The simulation relation is defined in terms of the mediator function, which maps runs to signals.

$$\begin{array}{ll}
\pi_{01}(t) & \equiv \llparenthesis\ \mathit{sigs} = r2s(t.\mathit{runs})\ \rrparenthesis \\
R_{01} & \equiv \{(s, t) \mid s = \pi_{01}(t)\}
\end{array}$$

At this point, we can state and prove the refinement result, which establishes **Requirement 2**.

Proposition 5. $\mathit{a1} \sqsubseteq_{R_{01},\pi_{01}} \mathit{a0}$.

We have now satisfied all system requirements: injective agreement between an initiator and a responder. By using abstraction we have captured the essential features of entity authentication protocols and established their main property, once and for all. Our proofs avoid the intricacies of an active attacker controlling all communication. However, the resulting model is too abstract and requires further refinement to become executable in the intended hostile distributed environment. The high level of abstraction allows different realizations, including the ISO/IEC 9798-3 (discussed below) and NSL/2 protocols.

4.4 Protocol using authentic channels (L2)

We now model the following abstract protocol using authentic channels.

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

The initiator A sends the nonce *Na* to B, who returns it together with his own nonce *Nb* on an authentic channel. We also introduce an explicit intruder who may fake messages as far as allowed by the channel's security properties. In the next section, we will refine this protocol into the ISO/IEC 9798-3 two-pass unilateral authentication protocol [18].

First, we define the type of channel messages for this protocol (cf. Section 2.2).

$$\begin{array}{ll}
\textbf{datatype}\ \mathit{imsg} & = \mathsf{M1}(\mathit{agent} \times \mathit{agent} \times \mathit{nonce}) \\
\textbf{datatype}\ \mathit{amsg} & = \mathsf{M2}(\mathit{nonce} \times \mathit{nonce}) \\
\textbf{datatype}\ \mathit{chmsg} & = \mathsf{Insec}(\mathit{imsg}) \\
 & \quad \mid \mathsf{Auth}(\mathit{agent} \times \mathit{agent} \times \mathit{amsg})
\end{array}$$

The types *imsg* and *amsg* define the payload message format for the insecure and authentic channels. For example, $\mathsf{Auth}(B, A, \mathsf{M2}(Nb, Na))$ denotes M2 containing the nonces Nb and Na, sent authentically from B to A.

We now extend the previous model's state with a new variable, *chan*, modeling a set of channel messages.

$$\textbf{record}\ \mathit{a2\_state} = \mathit{a1\_state}\ +\ \mathit{chan} : \mathit{chmsg\ set}$$

In the initial state, both fields are empty. The observation function projects this state to the field *runs*.

The protocol events send and receive messages to and from the insecure and authentic channels. In the second step, the responder B creates a new run identified by Nb, receives the message M1 from the insecure channel, and authentically sends the message M2 to A. This event refines the event *a1_step2* by adding the receiving and sending of messages. In particular, instead of accepting any nonce Na, the nonce is now extracted from M1.

$$\begin{array}{ll}
\mathit{a2\_step2}(A, B, Na, Nb) \equiv \{(s, s') \mid & \texttt{-- by}\ B \\
\quad Nb \notin \mathit{dom}(s.\mathit{runs}) \wedge & \texttt{-- fresh}\ Nb \\
\quad \mathsf{Insec}(\mathsf{M1}(A, B, Na)) \in s.\mathit{chan} \wedge & \texttt{-- recv M1} \\
\quad s'.\mathit{runs} := s.\mathit{runs}(Nb \mapsto (A, B, \mathsf{Resp}(Na))) \wedge & \\
\quad s'.\mathit{chan} := s.\mathit{chan}\ \cup\ \{\mathsf{Auth}(B, A, \mathsf{M2}(Nb, Na))\}\ \} &
\end{array}$$

In the third step, the initiator run Na receives the message M2 and updates its local state with the nonce Nb.

$$\begin{array}{ll}
\mathit{a2\_step3}(A, B, Na, Nb) \equiv \{(s, s') \mid & \texttt{-- by}\ A \\
\quad s.\mathit{runs}(Na) = (A, B, \mathsf{Init}(\bot)) \wedge & \\
\quad \mathsf{Auth}(B, A, \mathsf{M2}(Nb, Na)) \in s.\mathit{chan} \wedge & \texttt{-- recv M2} \\
\quad s'.\mathit{runs} := s.\mathit{runs}(Na \mapsto (A, B, \mathsf{Init}(Nb)))\ \} &
\end{array}$$

The reception of message M2 replaces the authentication guard with its direct access to the responder run's memory from the refined event *a1_step3*. The corresponding guard refinement **(GRD)** of the refinement proof requires the following invariant, which states that authentic messages between honest agents indeed originate from the associated responder run identified by Nb.

$$\begin{array}{l}
\mathit{a2\_inv1\_auth} \equiv \{s \mid \forall A, B, Na, Nb. \\
\quad \mathsf{Auth}(B, A, \mathsf{M2}(Nb, Na)) \in s.\mathit{chan} \wedge B \notin \mathit{bad} \wedge A \notin \mathit{bad} \\
\quad \rightarrow s.\mathit{runs}(Nb) = (A, B, \mathsf{Resp}(Na))\}
\end{array}$$

This invariant is all that is needed to prove the refinement.

We now define an intruder event for faking protocol messages, which refines *skip*. For the protocols we consider, this event is based on the intruder's key knowledge, denoted by $\mathit{ikk}(s.\mathit{chan})$, and his nonce knowledge, $\mathit{ink}(s.\mathit{chan})$. Since no keys are generated here, only the latter set is relevant: $\mathit{ink}(H)$ contains all nonces appearing in channel messages in H. We represent the set of messages that the intruder can fake by the term $\mathit{fakeable}(s.\mathit{chan})$, where $\mathit{fakeable}(H)$ is inductively defined by the following rules.[1]

$$\frac{M \in H}{M \in \mathit{fakeable}(H)} \qquad \frac{Na \in \mathit{ink}(H)}{\mathsf{Insec}(\mathsf{M1}(A, B, Na))) \in \mathit{fakeable}(H)}$$

$$\frac{B \in \mathit{bad} \vee A \in \mathit{bad} \quad Na \in \mathit{ink}(H) \quad Nb \in \mathit{ink}(H)}{\mathsf{Auth}(B, A, \mathsf{M2}(Nb, Na))) \in \mathit{fakeable}(H)}$$

The first rule covers the replay of an existing message. The other two rules state that the intruder can fake payloads with any agent names and nonces he knows. We assume that the intruder knows all agents' names. The first premise of the third rule restricts the faking of authentic messages to those with a dishonest sender or receiver (cf. Section 2.2).

The definition of the intruder event is now canonical: the intruder can send any fakeable message. We model this by closing the set of channel messages under fakeable messages.

$$\mathit{a2\_fake} \equiv \{(s, s') \mid s'.\mathit{chan} := \mathit{fakeable}(s.\mathit{chan})\}$$

Since this event only modifies the new variable *chan*, it refines *skip*.

Let *a2* be the above specification. The simulation relation for this refinement is defined by $R_{12} \equiv \Pi_{runs}$ (Section 2.1).

PROPOSITION 6. *Let* $R'_{12} = R_{12} \cap (\mathcal{U} \times \mathit{a2\_inv1\_auth})$. *Then* $\mathit{reach}(a2) \subseteq \mathit{a2\_inv1\_auth}$ *and* $a2 \sqsubseteq_{R'_{12},id} a1$.

Our model now includes an intruder acting in a distributed environment. By using abstract channels, we retain the possibility of different cryptographic realizations; e.g., we may realize the authentic channels using signatures or MACs.

4.5 ISO/IEC 9798-3 protocol (L3)

We now refine the abstract protocol model *a2* into the ISO/IEC 9798-3 two-pass unilateral authentication protocol [18] by translating the authentic channels into signed messages communicated over an insecure channel.

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

We also refine the intruder to a standard Dolev-Yao intruder.

The state *iso3_state* extends the first refinement's state with a set of messages *IK* modeling the intruder knowledge. Alternatively, *IK* can be seen as the network or a communication channel. This channel is insecure since the intruder can eavesdrop and inject messages.

$$\textbf{record}\ \mathit{iso3\_state} = \mathit{a1\_state}\ +\ \mathit{IK} : \mathit{msg\ set}$$

Initially, the field *runs* is empty and *IK* reflects a standard key distribution: the intruder knows all agents' public keys and the private keys of dishonest agents. This models **Assumptions 2** and **3**. Only the *runs* field is observable.

We restrict our presentation to the second protocol step. It refines the abstract event *a2_step2* by replacing the insecure message M1 by a corresponding message in *IK* and the authentic message M2 by a signed message in *IK*.

$$\begin{array}{ll}
\mathit{iso3\_step2}(A, B, Na, Nb) \equiv \{(s, s') \mid & \\
\quad Nb \notin \mathit{dom}(s.\mathit{runs}) \wedge & \\
\quad \{\!|\, A, B, Na \,|\!\} \in s.\mathit{IK} \wedge & \texttt{-- rcv M1} \\
\quad s'.\mathit{runs} := s.\mathit{runs}(Nb \mapsto (A, B, \mathsf{Resp}(Na))) \wedge & \\
\quad s'.\mathit{IK} := s.\mathit{IK}\ \cup\ \{\ \{\!|\, Nb, Na, A \,|\!\}_{pri(B)}\ \}\ \} & \texttt{-- send M2}
\end{array}$$

We now define the intruder with full Dolev-Yao capabilities: he can send any message in $\mathit{synth}(\mathit{analz}(s.\mathit{IK}))$. We model this by closing the intruder knowledge $s.\mathit{IK}$ under *synth* and *analz*.

$$\mathit{iso3\_DY\_fake} \equiv \{(s, s') \mid s'.\mathit{IK} := \mathit{synth}(\mathit{analz}(s.\mathit{IK}))\}$$

This event refines the abstract event *a2_fake*.

Let *iso3* be the above specification. This specification accounts for **Assumption 1**. Namely, the intruder knowledge

[1] For readability, we sometimes define sets by inductive rules. An equivalent definition using set comprehension is possible.

IK acts as the network for the protocol messages, and the intruder can fake messages in $synth(analz(s.IK))$.

The simulation relation in this refinement states that the intruder knowledge data-refines the channel messages of $a2$. We define the function abs_msg mapping a set of concrete messages to set of channel messages as follows.

$$\frac{\{\!| A, B, Na |\!\} \in H}{\mathsf{Insec}(\mathsf{M1}(A, B, Na)) \in abs\_msg(H)}$$

$$\frac{\{\!| Nb, Na, A |\!\}_{pri(B)} \in H}{\mathsf{Auth}(B, A, \mathsf{M2}(Nb, Na)) \in abs\_msg(H)}$$

The simulation relation R_{23} for the refinement proof is the intersection of the following two relations with Π_{runs}.

$$\begin{aligned} R_{23}^{msgs} &\equiv \{(s,t) \mid abs\_msg(parts(t.IK)) \subseteq s.chan\} \\ R_{23}^{non} &\equiv \{(s,t) \mid nonces(t.IK) \subseteq ink(s.chan)\} \end{aligned}$$

R_{23}^{msgs} states that the abstraction of the parts of the concrete messages is included in the abstract channel messages. R_{23}^{non} states that the nonces known to the concrete intruder, namely $nonces(t.IK) \equiv analz(t.IK) \cap ran(\mathsf{Nonce})$, are also known to the abstract intruder. If we prohibited the intruder from sending messages on an authentic channel for which he is the receiver (by removing the disjunct $A \in bad$ in the third rule defining $fakeable$), we could establish equalities instead of set inclusions in R_{23}^{msg} and R_{23}^{non}. However, the current definition makes the abstract intruder more powerful and enables alternative implementations of $a2$ based on symmetric cryptography, for instance, using MACs (cf. Section 2.2).

The refinement proof only requires a single internal invariant expressing the secrecy of the signing keys, namely, that $pri(A) \in analz(s.IK)$ implies $A \in bad$.

Proposition 7. *Let $R'_{23} = R_{23} \cap (\mathcal{U} \times iso3\_inv1\_keys)$. Then $reach(iso3) \subseteq iso3\_inv1\_keys$ and $iso3 \sqsubseteq_{R'_{23},id} a2$.*

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

$$\begin{aligned} &(s(\!| chan := fakeable(s.chan) |\!), \\ &\ t(\!| IK \quad := synth(analz(t.IK)) |\!)) \in R'_{23} \end{aligned}$$

This states that the successor states resulting from the respective intruder actions are still in the simulation relation. The part concerning R_{23}^{msgs} is proved using the following lemma.

Lemma 1. *Suppose $(s,t) \in R'_{23}$. Then*

$$abs\_msg(parts(synth(analz(t.IK)))) \subseteq fakeable(s.chan)$$

This lemma follows from the definitions of abs_msg and $fakeable$ and general properties of $parts$, $synth$, and $analz$.

Discussion. The above example illustrates our modeling principles and refinement strategy on a concrete example of an authentication protocol. We satisfied all system requirements by proving properties of the abstract models at Levels 0 and 1. As these models are not directly implementable, we continued our refinements, obtaining a final model that is suitable for implementation in the intended hostile distributed environment and, crucially, inherits the properties we proved for the abstract models. The simulation relation and invariants used here at Levels 2 and 3 are canonical for our refinement strategy (cf. Section 5).

5. KEY TRANSPORT PROTOCOLS

In this section, we first present an abstract, protocol-independent model of secrecy, called $s0$. We then instantiate the generic secret data in this model to keys and refine the resulting model into a concrete, server-based protocol: the Otway-Rees protocol as modified by Abadi and Needham [3] (called OR/AN). At Level 1 of our refinement strategy, we prove session key secrecy by refining the model $s0$ and we establish authentication properties by refining $a0i$ (from Section 4.2). At Level 2, the server directly sends the key on a secure channel to each role, reflecting the essential security and communication structure underlying server-based key transport protocols. The final refinement at Level 3 changes the communication topology. Here, the initiator forwards to the responder a ticket with the encrypted session key.

We have also derived the Needham-Schroeder Shared-Key (NSSK) protocol [29], the Yahalom protocol, and the core of Kerberos IV and V. These protocols also involve ticket forwarding. In the NSSK protocol, the responder's key is doubly encrypted in the server's message to the initiator. Moreover, these protocols employ the session key in their final message(s) to provide key confirmation guarantees.

5.1 Requirements and assumptions

Our requirements and assumptions are as follows.

Requirement 1 (Key distribution). The protocol generates and distributes a fresh session key.

Requirement 2 (Key secrecy). Only authorized agents may learn a session key, unless one of them is dishonest (whereby other dishonest agents may also learn it).

Requirement 3 (Authentication). Each recipient of the session key injectively agrees with the key generator on the key and possibly on additional data.

The environment assumptions, concerning the intruder and the cryptographic setup, are identical to those in Section 4.1.

5.2 Secrecy (L0)

We start our development with an abstract model of secrecy. We introduce two state variables, both relations between data (of polymorphic type δ) and agents: (i) kn, where $(d, A) \in s.kn$ means that agent A knows data d in state s, and (ii) az, where $(d, A) \in s.az$ means that A is authorized to know d in state s.

$$\textbf{record}\ \delta\ s0\_state = kn, az : (\delta \times agent)\ set$$

Secrecy can now be expressed as the following invariant.

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

Namely, all knowledge is authorized. We define the set of initial states to be those satisfying this invariant. The entire state is observable.

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

$$\begin{array}{ll} s0\_gen(d, A, G) \equiv \{(s, s') \mid & \\ \quad d \notin dom(s.kn) \wedge & \texttt{-- } d \texttt{ fresh} \\ \quad A \in G \wedge & \texttt{-- } A \texttt{ in group} \\ \quad s'.kn := s.kn \cup \{(d, A)\} \wedge & \\ \quad s'.az := s.az \cup \{(d, B) \mid B \in G \vee G \cap bad \neq \emptyset\}\ \} & \end{array}$$

The first guard requires that d is fresh, that is, not known to anybody. The second guard ensures that A is a member of G. There are two actions. The first adds the pair (d, A) to kn. The second updates the relation az with $\{d\} \times G$ if all members of G are honest and with $\{d\} \times \mathcal{U}_{agent}$ otherwise. In the latter case, there is no point in trying to achieve secrecy and therefore everyone is allowed to know d.

In the secret-learning event, an agent B learns the secret d. We require that someone knows d (first guard) and that B is authorized to learn d (second guard).

$$\begin{array}{ll}
\mathit{s0\_learn}(d, B) \equiv \{(s, s') \mid & \\
\quad d \in \mathit{dom}(s.kn) \wedge & \texttt{-- } d \texttt{ known to somebody} \\
\quad (d, B) \in s.az \wedge & \texttt{-- } B \texttt{ allowed to learn } d \\
\quad s'.kn := s.kn \cup \{(d, B)\} \, \} &
\end{array}$$

From a secrecy perspective, it is irrelevant from whom B learns d. Such aspects will be covered by authentication properties. The initial model *s0* clearly satisfies the secrecy invariant.

PROPOSITION 8. *Let s0 be the above specification. Then* $\mathit{reach}(\mathit{s0}) \subseteq \mathit{secrecy}$.

5.3 Abstract server-based key transport (L1)

We now define abstract *server-based* key transport protocols. We construct two models within Level 1, *kt1* and *kt1′*, and prove four refinements to establish secrecy (by refining *s0*) and authentication (by refining *a0i* from Section 4.2). These refinements are given in Propositions 9, 10, and 11 below and are summarized as follows.

$$\begin{array}{lllll}
\mathit{kt1}' & \sqsubseteq_{\pi_{11}} & \mathit{kt1} & \sqsubseteq_{\pi_{01}} & \mathit{s0} \\
\mathit{kt1}' & \sqsubseteq_{\pi_{01}^{ia}} & \mathit{a0i} & & \\
\mathit{kt1}' & \sqsubseteq_{\pi_{01}^{ra}} & \mathit{a0i} & &
\end{array}$$

Note that we obtain a refinement graph rather than a linear sequence of refinements.

The first model, *kt1*, refines the secrecy model *s0* to server-based secret key distribution, thus establishing **Requirements 1** & **2**. This model introduces a local run state for each role. Neither communication channels nor intruder events exist in this model. Instead, the initiator and the responder read the session key generated by the server from the server's memory and the intruder is implicit in the events' guards. The model *kt1* is a common ancestor in the refinement graph of all server-based key transport protocols that we have derived.

The second model, *kt1′*, realizes the authentication properties of **Requirement 3**, both for the initiator and for the responder, by using nonces. This model trivially refines *kt1* by adding an authentication guard to each client role's key-reading event in *kt1*. We prove a separate refinement of the authentication model *a0i* for each of the two authentication properties, each with a different mediator function and simulation relation. The key transport protocols we have derived differ with respect to their authentication guarantees. Hence, their models at this level require the addition of different authentication guards to *kt1*.

We assume an initial key setup, described by an uninterpreted relation knC_0 between keys and agents. This relation will be defined later on Level 3. We call the keys in this relation's domain static or long-term keys. Corrupted keys are static keys known by some bad agent.

$$\begin{array}{ll}
\mathit{staticKey} \equiv \mathit{dom}(knC_0) & \texttt{-- long-term keys} \\
\mathit{corrKey} \equiv knC_0^{-1}[\mathit{bad}] & \texttt{-- corrupted keys}
\end{array}$$

5.3.1 Secret key distribution

At Level 1, we use the following definition of state for all server-based key distribution protocols.

$$\begin{array}{ll}
\textbf{datatype } \mathit{frame} = & \mathsf{Init}(\mathit{key}_\perp \times \mathit{nonce\ list}) \\
& \mid \mathsf{Resp}(\mathit{key}_\perp \times \mathit{nonce\ list}) \\
& \mid \mathsf{Serv}(\mathit{nonce\ list}) \\
\textbf{record } \mathit{kt1\_state} = & \mathit{runs} : \mathit{rid} \rightharpoonup \mathit{agent} \times \mathit{agent} \times \mathit{frame}
\end{array}$$

The frame records the role-specific information. The initiator and the responder record the session key and all three roles store a list of nonces. Nonces will only be recorded (if at all) in the second model *kt1′*. The definition of a common state for both *kt1* and *kt1′* is a modeling decision that slightly simplifies specifications and proofs rather than a necessity. We use client run identifiers as nonces and server run identifiers as session keys. Initially the *runs* map is empty. The entire state is observable.

For the refinement, we define the functions knC and azC, which reconstruct the knowledge and authorization relations, kn and az, of the initial model *s0* from the runs. The function knC is defined by the following rules, where r is a variable of the same type as *s.runs*.

$$\frac{r(N) \in \{(A, B, \mathsf{Init}(K, ns)), (B, A, \mathsf{Resp}(K, ns))\}}{(K, A) \in knC(r)}$$

$$\frac{r(K) = (A, B, \mathsf{Serv}(ns))}{(K, \mathsf{S}) \in knC(r)} \qquad \frac{(K, C) \in knC_0}{(K, C) \in knC(r)}$$

The first two rules describe session key knowledge for each role. The third rule includes initially known keys. The second function, azC, is defined as follows.

$$\frac{r(K) = (A, B, \mathsf{Serv}(ns)) \quad C \in \{A, B, \mathsf{S}\}}{(K, C) \in azC(r)}$$

$$\frac{r(K) = (A, B, \mathsf{Serv}(ns)) \quad \{A, B\} \cap \mathit{bad} \neq \emptyset}{(K, C) \in azC(r)}$$

$$\frac{(K, C) \in knC_0}{(K, C) \in azC(r)} \qquad \frac{K \in \mathit{corrKey}}{(K, C) \in azC(r)}$$

By the first rule A, B, and the server S are authorized to know the key K that the server generated for A and B. The second rule expresses that everyone may learn session keys that the server generated for some dishonest agent A or B. The last two rules state that $azC(r)$ includes the initial key setup and that everyone is authorized to learn corrupted keys.

The specification *kt1* consists of five events. The first event is new and thus refines *skip*. It creates a new run (and nonce) Na of initiator A with responder B by updating *runs* with $(Na \mapsto (A, B, \mathsf{Init}(\perp))$. The second event creates a responder run analogously. In the third event, which refines *s0_gen*, the server generates a fresh session key Kab and records it together with the agent names A and B in *runs*.

$$\begin{array}{ll}
\mathit{kt1\_step3}(A, B, Kab) \equiv \{(s, s') \mid & \texttt{-- by } \mathsf{S} \\
\quad Kab \notin \mathit{dom}(s.runs) \cup \mathit{staticKey} \wedge & \\
\quad s'.runs := (s.runs)(Kab \mapsto (A, B, \mathsf{Serv}([]))) \, \} &
\end{array}$$

Guard strengthening for this event requires an auxiliary invariant, *kt1_inv1_key*, stating that each key in $knC(s.runs)$ is either a long-term key or identifies a run.

The final two events both refine the event *s0_learn*. In the following event, the responder B acquires the session key Kab from S by directly reading the server's memory.

$$\begin{array}{ll}
\mathit{kt1\_step4}(A, B, \mathit{Nb}, \mathit{Kab}) \equiv \{(s, s') \mid & \texttt{-- by } B \\
\quad s.\mathit{runs}(\mathit{Nb}) = (A, B, \mathsf{Resp}(\bot, [])) \wedge & \\
\quad \mathit{Kab} \in \mathit{sessionKeys}(s) \cup \mathit{corrKey} \wedge & \texttt{-- } \mathit{Kab} \texttt{ known} \\
\quad (\mathit{Kab}, B) \in \mathit{azC}(s) \wedge & \texttt{-- check authorization} \\
\quad s'.\mathit{runs} := (s.\mathit{runs})(\mathit{Nb} \mapsto (A, B, \mathsf{Resp}(\mathit{Kab}, []))) \; \} &
\end{array}$$

The first guard requires that Nb identifies a run by responder B with initiator A that has not yet received a key. The second guard states that Kab is an existing session key (that is, identifies a server run) or a corrupted key. This guard strengthens the first guard in *s0_learn* expressing that Kab is known by someone. The inclusion of corrupted keys reflects that the Dolev-Yao intruder at Level 3 may send such keys to dishonest agents. This does not violate session key secrecy. The authentication properties realized by $\mathit{kt1}'$ will prevent that honest agents accept corrupted keys. The third guard ensures that B is authorized to learn Kab, expressed using the relation azC defined above. The action updates the responder run with the session key Kab taken from S's memory. The event *kt1_step5* performs an analogous step for the initiator.

We establish that the model *kt1* defined above is a data refinement of the model *s0* from Section 5.2. The mediator function π_{01} and the simulation relation R_{01} abstract *kt1*'s states to *s0*'s using the functions knC and azC.

$$\begin{array}{ll}
\pi_{01}(t) & \equiv (\!| \; \mathit{kn} = \mathit{knC}(t.\mathit{runs}), \mathit{az} = \mathit{azC}(t.\mathit{runs}) \; |\!) \\
R_{01} & \equiv \{(s, t) \mid s = \pi_{01}(t)\}
\end{array}$$

We can now prove the following refinement result.

Proposition 9. *Let* $S_{01} = \mathcal{U} \times \mathit{kt1\_inv1\_key}$. *Then we have* $\mathit{reach}(\mathit{kt1}) \subseteq \mathit{kt1\_inv1\_key}$ *and* $\mathit{kt1} \sqsubseteq_{R_{01} \cap S_{01}, \pi_{01}} \mathit{s0}$.

Since the abstract variables kn and az are observable and can be reconstructed from the concrete state, the secrecy invariant for *s0* (Proposition 8) is also fulfilled by the specification *kt1* (by Propositions 1 and 3), which therefore realizes **Requirements 1** & **2**.

5.3.2 Authenticated key distribution

Although the model *kt1* provides secret key distribution, there is no guarantee of key freshness: the same session key may be distributed many times and, worse, initiator and responder may even obtain corrupted keys. We address these issues by using nonces to prevent replays and guarantee key freshness as is standard. We use the initiator and responder's run identifiers, Na and Nb, as nonces, add authentication guards to the events of *kt1*, and refine the model *a0i* once for each client role to establish an injective agreement with the server on the client's nonce, the other client's name, and the session key.

Concretely, we transform the model *kt1* into $\mathit{kt1}'$. First, we add two nonce parameters, Na and Nb, to *kt1_step3* and modify its action to record the list $[\mathit{Na}, \mathit{Nb}]$ in the server state. The nonces are arbitrary since their origin is unclear. Indeed, the clients will later send these nonces in the clear and the intruder may thus replace them by others.

Second, for the responder B to agree with the server S on $d = (\mathit{Kab}, A, \mathit{Nb})$, we add an authentication guard to B's event *kt1_step4*. This guard states that the server has generated Kab for A and B and knows Nb, provided B is honest.

$$B \notin \mathit{bad} \rightarrow \exists \mathit{Na}.\; s.\mathit{runs}(\mathit{Kab}) = (A, B, \mathsf{Serv}([\mathit{Na}, \mathit{Nb}]))$$

The initiator's nonce Na in the server's frame is bound by the existential quantifier, since it is not part of the data d.

Finally, we add a similar authentication guard to the event *kt1_step5* for the initiator A to agree with the server S on $d = (\mathit{Kab}, B, \mathit{Na})$. This guard assumes that A is honest and quantifies over the nonce Nb in the server's frame.

It is easy to see that $\mathit{kt1}'$ refines *kt1*. The mediator function π_{11} and the simulation relation R_{11} between *kt1* and $\mathit{kt1}'$ map the lists of nonces in the run frames to the empty list, but do not otherwise affect the runs.

Proposition 10. $\mathit{kt1}' \sqsubseteq_{R_{11}, \pi_{11}} \mathit{kt1}$.

We now examine the two refinements of *a0i*, each of which establishes an injective agreement between one client role (the initiator or responder) and the server. Since these two refinements are similar, we restrict our discussion to the responder case, where we want the responder B to injectively agree with the server S on $d = (\mathit{Kab}, A, \mathit{Nb})$. The simulation relation maps completed server runs to $\mathsf{Running}$ signals and completed responder runs to Commit signals as follows.

$$r2s(r)(S) = \begin{cases} 1 & \text{if } S = \mathsf{Commit}([B, \mathsf{S}], (\mathit{Kab}, A, \mathit{Nb})) \\ & \text{and } r(\mathit{Nb}) = (A, B, \mathsf{Resp}(\mathit{Kab})) \\ 1 & \text{if } S = \mathsf{Running}([B, \mathsf{S}], (\mathit{Kab}, A, \mathit{Nb})) \\ & \text{and } r(\mathit{Kab}) = (A, B, \mathsf{Serv}([\mathit{Na}, \mathit{Nb}])) \\ 0 & \text{otherwise} \end{cases}$$

For this agreement, we need not assume that A is honest. Therefore, A appears as part of the data $d = (\mathit{Kab}, A, \mathit{Nb})$ rather than in the list of agents $h = [B, \mathsf{S}]$. We include S in h for uniformity, but we could remove it, since S is a fixed honest agent by a global assumption.

The mediator function π^{ra}_{01} and simulation relation R^{ra}_{01} are defined such that $(s, t) \in R^{ra}_{01}$ and $s = \pi^{ra}_{01}(t)$ are equivalent to $s.\mathit{sigs} = r2s(t.\mathit{runs})$ (cf. Section 4.3). Given these definitions, the server event $\mathit{kt1\_step3}(A, B, \mathit{Na}, \mathit{Nb}, \mathit{Kab})$ refines the abstract event $\mathit{a0i\_running}([B], (\mathit{Kab}, A, \mathit{Nb}))$ and the responder event $\mathit{kt1\_step4}(A, B, \mathit{Nb}, \mathit{Kab})$ refines the abstract event $\mathit{a0i\_commit}([B], (\mathit{Kab}, A, \mathit{Nb}))$. The remaining events refine *skip*.

Let π^{ia}_{01} and R^{ia}_{01} be the analogous mediator function and simulation relation defined for the initiator.

Proposition 11. *The following refinements hold:*

$$\mathit{kt1}' \sqsubseteq_{R^{ia}_{01}, \pi^{ia}_{01}} \mathit{a0} \qquad \textit{and} \qquad \mathit{kt1}' \sqsubseteq_{R^{ra}_{01}, \pi^{ra}_{01}} \mathit{a0}.$$

This proposition establishes **Requirement 3**. The proofs do not require auxiliary invariants. We have now satisfied all system requirements. However, further refinements are needed to realize the environment assumptions and thus obtain a protocol that is executable in the intended hostile network environment.

5.4 Protocol using secure channels (L2)

We now introduce an abstract key transport protocol using secure channels and an explicit intruder event. The protocol reads as follows.

$$\begin{array}{llcl}
\text{M1.} & A \rightarrow B & : & A, B, \mathit{Na} \\
\text{M2.} & B \rightarrow \mathsf{S} & : & A, B, \mathit{Na}, \mathit{Nb} \\
\text{M3.} & \mathsf{S} \bullet\!\!\rightarrow\!\!\bullet B & : & \mathit{Nb}, A, \mathit{Kab} \\
\text{M4.} & \mathsf{S} \bullet\!\!\rightarrow\!\!\bullet A & : & \mathit{Na}, B, \mathit{Kab}
\end{array}$$

The initiator A begins by sending a nonce Na to the responder B (M1). Then B adds his nonce Nb and sends it to S

(M2). The server generates a session key Kab and sends it on secure channels to B (M3) and A (M4) together with their respective nonces.

The state of the model *kt2* extends the previous model's state with a set of *channel messages*. There are insecure and secure channel messages, where for the secure ones, the sender S is fixed and thus dropped from the constructor (cf. Section 2.2). The type of insecure payload, *imsg* (not shown below), covers the first two protocol messages. The type of secure payload, *smsg*, covers messages M3 and M4. In the initial state, *chmsg* and the run maps are empty.

$$\begin{array}{lll}
\textbf{datatype}\ smsg & = & \mathsf{M3}(nonce \times agent \times key) \\
 & \mid & \mathsf{M4}(nonce \times agent \times key) \\
\textbf{datatype}\ chmsg & = & \mathsf{Insec}(imsg) \mid \mathsf{Secure}(agent \times smsg) \\
\textbf{record}\ kt2\_state & = & kt1\_state\ +\ chan\ :\ chmsg\ set
\end{array}$$

The function *kt2_obs* allows the observation of the run fields and the intruder key knowledge $ikk(s.chan)$. The set $ikk(H)$ contains the keys in secure channels with a dishonest receiver and the long-term keys of dishonest agents. The rule for M4 is similar to the one for M3 and is not shown.

$$\frac{\mathsf{Secure}(C, \mathsf{M3}(N, A, K)) \in H \quad C \in bad}{K \in ikk(H)} \qquad \frac{K \in corrKey}{K \in ikk(H)}$$

Observing the keys suffices to state session-key secrecy in terms of *ikk* as an observable (and thus preserved) invariant.

The model *kt2* has five protocol events and a new intruder event. Each protocol event refines the corresponding event of the previous model. The event *kt2_step1* achieves this by sending M1 on the insecure channel. The event *kt2_step2* models the responder receiving M1 and sending M2 on the insecure channel. The server event *kt2_step3* receives message M2 on the insecure channel and simultaneously sends the secure messages M3 and M4 to B and A.

$$\begin{array}{ll}
kt2\_step3(A, B, Na, Nb, Kab) \equiv \{(s, s') \mid & \texttt{-- by S} \\
\quad Kab \notin dom(s.runs) \cup staticKey\ \wedge & \\
\quad \mathsf{Insec}(\mathsf{M2}(A, B, Na, Nb)) \in s.chan\ \wedge & \texttt{-- recv M2} \\
\quad s'.runs := (s.runs)(Kab \mapsto (A, B, \mathsf{Serv}([Na, Nb])))\ \wedge & \\
\quad s'.chan := s.chan\ \cup & \texttt{-- send M3, M4} \\
\qquad\qquad \{\mathsf{Secure}(B, \mathsf{M3}(Nb, A, Kab)), & \\
\qquad\qquad\ \ \mathsf{Secure}(A, \mathsf{M4}(Na, B, Kab))\}\ \} &
\end{array}$$

The event *kt2_step4* refines its abstract counterpart by replacing the authentication guard, involving direct access to the server's memory, with a guard modeling the responder B receiving message M3. The event *kt2_step5* analogously models the initiator A receiving M4.

$$\begin{array}{ll}
kt2\_step4(A, B, Nb, Kab) \equiv \{(s, s') \mid & \texttt{-- by}\ B \\
\quad s.runs(Nb) = (A, B, \mathsf{Resp}(\bot, []))\ \wedge & \\
\quad \mathsf{Secure}(B, \mathsf{M3}(Nb, A, Kab)) \in s.chan\ \wedge & \\
\quad s'.runs := (s.runs)(Nb \mapsto (A, B, \mathsf{Resp}(Kab, [])))\ \} &
\end{array}$$

The intruder event is canonical: it closes the set of channel messages under fakeable messages (Section 4.4). The intruder's knowledge in state s consists of the keys $ikk(s.chan)$, defined above, and the nonces $ink(s.chan)$. The set of nonces $ink(H)$ is defined using the function *non*, which extracts the set of nonces from a channel message, as follows.

$$\frac{\begin{array}{c}\mathsf{Insec}(M) \in H \\ N \in non(\mathsf{Insec}(M))\end{array}}{N \in ink(H)} \qquad \frac{\begin{array}{c}\mathsf{Secure}(C, M) \in H \quad C \in bad \\ N \in non(\mathsf{Secure}(C, M)))\end{array}}{N \in ink(H)}$$

We define the set $fakeable(H)$ analogously to Section 4.4. The rules for the secure messages allow the intruder to only fake messages with a dishonest receiver (cf. Section 2.2).

Let *kt2* be the above specification. The simulation relation $R_{12} \equiv \Pi_{runs}$ asserts that the runs in *kt1′* and *kt2* are identical. The mediator function $\pi_{12} \equiv \pi_{runs}$ removes the keys from *kt2*'s observations.

The guard strengthening **(GRD)** proof in the refinement of steps 4 and 5 is derived from two invariants relating the secure messages M3 and M4 with the server state. Both invariants contain two almost identical conjuncts of which we only discuss the one concerning M3. The first invariant states that the key K in message $\mathsf{M3}(N, A, K)$ sent to a dishonest agent B is a session key that the server generated for some dishonest agent or a corrupted key.

$$\begin{array}{l}
kt2\_inv1\_M34\_bad \equiv \{s \mid \forall A, B, Nb, K.\ \ \ldots\ \wedge \\
\quad \mathsf{Secure}(B, \mathsf{M3}(Nb, A, K)) \in s.chan\ \wedge\ B \in bad \\
\quad \rightarrow (\exists A', B', z.\ s.runs(K) = (A', B', \mathsf{Serv}(z))) \\
\qquad\qquad\qquad \wedge\ (A' \in bad \vee B' \in bad))\ \vee\ K \notin corrKey\,\}
\end{array}$$

The second invariant states that if a key Kab, agent A, and a nonce Nb appear in a secure message to an honest B then the server has generated the session key Kab for A and B and he knows the nonce Nb.

$$\begin{array}{l}
kt2\_inv2\_M34\_good \equiv \{s \mid \forall A, B, Nb, Kab.\ \ \ldots\ \wedge \\
\quad \mathsf{Secure}(B, \mathsf{M3}(Nb, A, Kab)) \in s.chan\ \wedge\ B \notin bad \\
\quad \rightarrow (\exists N.\ s.runs(Kab) = (A, B, \mathsf{Serv}(N, Nb)))\ \}
\end{array}$$

This invariant directly implies the refinement of the authentication guards of *kt1_step4* and *kt1_step5* by the guards modeling the clients receiving messages M3 and M4. The refinements of the other two guards of these two events require both invariants. We can now prove the following result.

Proposition 12. *Let kt2_invs be the intersection of the two invariants of kt2 and let* $R'_{12} \equiv R_{12}\ \cap\ (\mathcal{U} \times kt2\_invs)$. *Then* $reach(kt2) \subseteq kt2\_invs$ *and* $kt2 \sqsubseteq_{R'_{12}, \pi_{12}} kt1'$.

We proved three additional invariants stating key secrecy guarantees in terms of *ikk* for each of the roles. Here is the one concerning the server:

$$\begin{array}{l}
kt2\_inv4\_ikk\_sv \equiv \{s \mid \forall A, B, K, x. \\
\quad s.runs(K) = (A, B, \mathsf{Serv}(x))\ \wedge\ A \notin bad\ \wedge\ B \notin bad \\
\quad \rightarrow K \notin ikk(s.chan)\,\}
\end{array}$$

Since *ikk* is observable, these guarantees will be preserved by the next refinement.

5.5 Otway-Rees/AN protocol (L3)

In this section, we refine our abstract key transport protocol to the Otway-Rees/AN protocol [3]. The first two messages remain the same, while M3 and M4 are refined in two ways. First, they are implemented using symmetric encryption.[2] We assume that each agent A shares a symmetric key $shr(A)$ with the server S.

$$\begin{array}{lll}
\text{M3'.} & \mathsf{S} \rightarrow B: & \{\!|\, Na, A, B, Kab \,|\!\}_{shr(A)}, \\
 & & \{\!|\, Nb, A, B, Kab \,|\!\}_{shr(B)} \\
\text{M4'.} & B \rightarrow A: & \{\!|\, Na, A, B, Kab \,|\!\}_{shr(A)}
\end{array}$$

Hence, we concretize the initial key setup by defining the previously uninterpreted constant knC_0 as follows.

$$knC_0 \equiv \{(shr(A), B) \mid B = A \vee B = \mathsf{S}\}$$

[2] In contrast to cryptographic models, symmetric encryption in the Dolev-Yao model also guarantees authenticity.

From this definition we prove that $\mathit{staticKey} = \mathit{ran}(\mathit{shr})$ and $\mathit{corrKey} = \mathit{shr}[\mathit{bad}]$. Second, this refinement modifies the communication topology: The server encrypts the session key Kab for A and B, and sends the resulting message (M3') to B, who forwards A's ticket (M4'). Moreover, we refine the intruder fake event into a standard Dolev-Yao intruder.

The state of *or3* extends the record *kt1_state* with a field $\mathit{IK} : \mathit{msg\ set}$ for the intruder knowledge, thus refining the abstract channels *chan* of *kt2* to the intruder knowledge *IK* (cf. Section 4.5). Initially, the runs are empty and the intruder knows the dishonest agents' long-term keys. This formalizes **Assumptions 2** and **3**. The observations consist of the runs and the set $\mathit{keys}(s.\mathit{IK}) \equiv \mathit{analz}(s.\mathit{IK}) \cap \mathit{ran}(\mathsf{Key})$.

The first, second, and fifth protocol steps are straightforward refinements of their previous versions. In the third step, the server sends B a message consisting of a pair of ciphertexts containing a copy of the session key for A and B (M3') in response to B's request message (M2). This differs from the previous refinement, where the server sends the keys to A and B on two separate secure channels.

$$\begin{array}{ll}
\mathit{or3\_step3}(A, B, \mathit{Na}, \mathit{Nb}, \mathit{Kab}) \equiv \{(s, s') \mid & \texttt{-- by S} \\
\quad \mathit{Kab} \notin \mathit{dom}(s.\mathit{runs}) \cup \mathit{ran}(\mathit{shr}) \wedge & \\
\quad \{\!|\, A, B, \mathit{Na}, \mathit{Nb} \,|\!\} \in s.\mathit{IK} \wedge & \texttt{-- recv M2} \\
\quad A \neq B \wedge & \\
\quad s'.\mathit{runs} := s.\mathit{runs}(\mathit{Kab} \mapsto (A, B, \mathsf{Serv}(\mathit{Na}, \mathit{Nb}))) \wedge & \\
\quad s'.\mathit{IK} := s.\mathit{IK} \cup & \texttt{-- send M3'} \\
\quad\ \{ \{\!|\, \{\!|\, \mathit{Na}, A, B, \mathit{Kab} \,|\!\}_{\mathit{shr}(A)}, \{\!|\, \mathit{Nb}, A, B, \mathit{Kab} \,|\!\}_{\mathit{shr}(B)} \,|\!\} \} \} &
\end{array}$$

The guard $A \neq B$ ensures that we can distinguish the components of M3' and abstract them to M3 and M4. We could avoid this (mild) restriction by tagging these components.

In the fourth step, B receives his encrypted copy of the session key paired with an uninterpreted message X (called a ticket) that he forwards to A.

$$\begin{array}{ll}
\mathit{or3\_step4}(A, B, \mathit{Nb}, \mathit{Kab}, X) \equiv \{(s, s') \mid & \texttt{-- by } B \\
\quad s.\mathit{runs}(\mathit{Nb}) = (A, B, \mathsf{Resp}(\bot, [])) \wedge & \\
\quad \{\!|\, X, \{\!|\, \mathit{Nb}, A, B, \mathit{Kab} \,|\!\}_{\mathit{shr}(B)} \,|\!\} \in s.\mathit{IK} \wedge & \texttt{-- recv M3'} \\
\quad s'.\mathit{runs} := s.\mathit{runs}(\mathit{Nb} \mapsto (A, B, \mathsf{Resp}(\mathit{Kab}, []))) \wedge & \\
\quad s'.\mathit{IK} := s.\mathit{IK} \cup \{X\} \} & \texttt{-- forward } X
\end{array}$$

Let *or3* be the specification described above, including a Dolev-Yao intruder event as in Section 4.5. This event, together with the fact that the intruder knowledge contains all sent and received messages, accounts for **Assumption 1**.

The simulation relation R_{23} refines the channel messages *chan* to the intruder knowledge *IK*. It is the intersection of four relations: R_{23}^{msgs}, R_{23}^{keys}, R_{23}^{non}, and Π_{runs}. The first two of these are defined as follows.

$$\begin{array}{lcl}
R_{23}^{\mathit{msgs}} & \equiv & \{(s, t) \mid s.\mathit{chan} = \mathit{abs\_msg}(\mathit{parts}(t.\mathit{IK}))\} \\
R_{23}^{\mathit{non}} & \equiv & \{(s, t) \mid \mathit{ink}(s.\mathit{chan}) = \mathit{nonces}(t.\mathit{IK})\}
\end{array}$$

These two relations strengthen the (right-to-left) set inclusions appearing in their analogues from Section 4.5 to an equality. Of course, we must adapt the definition of the message abstraction function *abs_msg* to the Otway-Rees/AN protocol. This function abstracts the component messages of M3' separately. Here is the rule concerning M3; the one for M4 is symmetrical.

$$\frac{\{\!|\, \mathit{Nb}, A, B, \mathit{Kab} \,|\!\}_{\mathit{shr}(B)} \in H}{\mathsf{Secure}(B, \mathsf{M3}(\mathit{Kab}, A, \mathit{Nb})) \in \mathit{abs\_msg}(H)}$$

The relation R_{23}^{keys} states that the intruder key knowledge is identical in the abstract and concrete models. This equality is needed to prove that R_{23} respects observations mediated by the identity function.

$$R_{23}^{\mathit{keys}} \equiv \{(s, t) \mid \mathit{ikk}(s.\mathit{chan}) = \mathit{keys}(t.\mathit{IK})\}$$

The main invariant concerns the secrecy of the long-term keys: it expresses that $\mathit{shr}(A) \in \mathit{analz}(s.\mathit{IK})$ if and only if $A \in \mathit{bad}$. Its proof requires two additional auxiliary invariants concerning the definedness of the keys used for encryption and the keys carried by messages.

Proposition 13. *Let or3_invs be the intersection of all invariants of or3 and let* $R'_{23} = R_{23} \cap (\mathcal{U} \times \mathit{or3\_invs})$. *Then* $\mathit{reach}(\mathit{or3}) \subseteq \mathit{or3\_invs}$ *and* $\mathit{or3} \sqsubseteq_{R'_{23},\mathit{id}} \mathit{kt2}$.

Discussion. We have also developed direct cryptographic implementations of the Level 2 key transport protocols, that is, without forwarding. Remarkably, the proof scripts of the direct and the forwarding refinement of OR/AN differ only by a few lines. The same remark applies to the Yahalom protocol. The difference for NSSK is more significant. Due to nested encryption, four more invariants are required that describe the shape of the doubly encrypted messages. This reflects the clear separation of concerns in our refinement strategy: Level 2 deals with the security concerns stemming from a distributed environment and an active intruder, while Level 3 handles details concerning cryptography (double encryption) and communication topology (forwarding).

6. RELATED WORK

There have been other proposals for developing security protocols by refinement using different formalisms such as the B method [9], its combination with CSP [12], I/O automata [23], and ASMs [6]. None of these continue their refinements to the level of a full Dolev-Yao intruder. Either they only consider an intruder that is passive [23], defined ad-hoc [6, 12], or corresponds to our Level 2 intruder [9]. This makes a comparison of their results with standard protocol models difficult. Moreover, none of them proposes a uniform and systematic development method as we do with our four-level refinement strategy and most of them develop individual protocols rather than entire families.

Datta et al. [16] 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. [13,32] similarly refine protocols by transforming messages and propose specialized formalisms for establishing secrecy and authentication properties. These refinements do not involve a fundamental change of the abstraction level since one abstracts and instantiates operations on messages. In contrast, our refinements bridge four distinct abstraction levels. Initially, we consider neither protocol runs or messages nor an active intruder. As a consequence, the main security properties are easy to prove and are preserved by further refinements that introduce the intended hostile distributed environment.

Classical notions of refinement (such as simulation) do not preserve information-flow properties, since they involve a reduction of non-determinism, which can destroy secrecy. Several works address this problem, known as the refinement paradox, for example, [5, 19, 24, 26]. Morgan and McIver [26, 28] solve the paradox by explicitly recording the set of possible values of secret variables. These sets represent the intruder's ignorance and refinements may extend, but never reduce them.

For classical security protocols, Cortier et al. [14] show that strong secrecy is equivalent to reachability-based secrecy (used here), if the secrets are not tested. This condition holds for the session keys in the OR/AN protocol, but fails for protocols that involve key confirmation.

Abstract channels and their transformations were studied by Maurer and Schmid [25]. Boyd has formalized analogous results using Z [11]. Bieber et al. model abstract channels using the B method [10] and refine them to cryptographic implementations. Abadi et al. [1] formalize secure channels in a variant of the join calculus and establish full abstraction results for translations to cryptographic implementations.

Mödersheim and Viganò [27] have developed a security protocol model based on abstract channels (L2) and related it to a more standard Dolev-Yao model (L3). They show the equivalence of the two models under the restriction that all protocol messages are completely decryptable. They use a fixed translation of channel messages to (public-key) cryptographic messages. Since our models include tickets and allow different realizations of abstract channels, their result does not cover our refinements from L2 to L3.

Several authors address the problem of synthesizing secure protocols from high-level specifications. Bhargavan et al. [8] compile multi-party sessions into ML modules equipped with dependent types. Successful type checking provides strong integrity guarantees (matching conversations) for the honest agents in a session even in the presence of corrupted agents and an active intruder. Cortier et al. [15] translate a single-session protocol into a multi-session protocol secure against a Dolev-Yao intruder. As in our L1 models, the abstract models in both works do not include an intruder, which simplifies reasoning about them. Their translations assume a fixed (public-key) cryptographic setup, whereas our approach provides flexibility in adapting to different (for example, wireless) settings. Protocol synthesis in a cryptographic setting is studied, for instance, in [7, 20], which present security-preserving translations from protocols designed for authentic channels (that is, a passive intruder) to protocols operating on insecure channels controlled by an active intruder.

Paulson [31] uses induction to define the protocols' event traces and verify their (protocol-dependent) security properties post-hoc. Using refinement, we can express security properties in a protocol-independent way at L0 and establish them in the first refinement (L1) in the absence of an active attacker. Our L3 models use Paulson's Isabelle/HOL theory of cryptographic messages and the operators *parts*, *analz*, and *synth*. Usually, only a few properties remain to be proved at this level (for example, the secrecy of long-term keys).

7. CONCLUSIONS

We propose a refinement strategy for the systematic development of security protocols. The abstract models help the developer to focus on the essentials: In our case studies, we have established all requirements on abstract models containing neither communication channels nor intruder events. Our refinement strategy guides the developer towards the concrete levels that account for the environment assumptions, namely, the distributed environment controlled by a Dolev-Yao intruder.

Compared to many post-hoc verification methods, including Paulson's, our development process scales better to more complex protocols. First, starting from abstract models and adding details in an incremental manner frequently leads to simpler solutions than one-shot designs. For example, the unnecessary double-encryption in Kerberos IV would be avoided using our approach. Second, secrecy and authentication properties are formulated independently of the protocol (at Level 0) and proved at much higher levels of abstraction (Level 1), although subsequent refinement proofs are required for the preservation of these properties. Third, the levels of our refinement strategy are reflected in well-structured proofs of correctness, where each level comes with its own particular invariants and simulation relations, many of which are canonical. In contrast, in post-hoc verification one is immediately confronted with much more complex state spaces, which makes discovering and proving suitable invariants more difficult.

The abstract channel model is very simple and protocol messages with nested cryptographic operations or undecryptable message parts have no direct representation. This excludes representing, for example, the forwarding of undecryptable messages (OR/AN, Yahalom, and NSSK), messages containing certificates, and nested encryption (NSSK). Our experience has convinced us that this simplicity is a virtue rather than a limitation. This model naturally reflects the actual (star-shaped) security architecture of server-based key transport protocols and to view forwarding and double encryption as implementation techniques, to be dealt with at the final level. Our developments show that this is possible. Such abstractions are even more beneficial for developing new protocols. From this perspective, certificates provide an abstract authentic channel from the certification authority to the agent verifying it and encrypted and signed messages are just one way of implementing a secure channel. These features, and many others, are within the scope of our method. For example, we have also derived the NSSK, Yahalom, and core Kerberos protocols, in which key confirmation is achieved by exchanging nonces or timestamps encrypted with the session key. These protocols require a straightforward extension of the channel model presented here with dynamic channels indexed by keys.

There are many possibilities for future work. We plan to improve the automation of both model and proof construction. The models could be synthesized in part from high-level specifications of security properties. Proof automation could be improved by automatically generating invariants and refinements, taking full advantage of the structure of our refinement process, and by proving them using the various automatic theorem provers working with Isabelle. Moreover, we plan to further enhance the scalability of our approach by developing additional infrastructure to support abstract channels and by supporting compositional reasoning about protocols. Finally, we want to apply our method to other classes of protocols, such as key agreement and multi-party protocols, and prove more complex properties such as security under various forms of key compromise.

Acknowledgements

We thank Jean-Raymond Abrial, Son Thai Hoang, and Egon Börger for fruitful discussions. We are also grateful to Cas Cremers, Simon Meier, Benedikt Schmidt, Patrick Schaller, and the anonymous reviewers for their helpful comments.

8. REFERENCES

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

[2] M. Abadi and L. Lamport. The existence of refinement mappings. *Theor. Comput. Sci.*, 82(2):253–284, 1991.

[3] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Softw. Eng.*, 22(1):6–15, 1996.

[4] J.-R. Abrial and S. Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundam. Inform.*, 77(1-2):1–28, 2007.

[5] R. Alur, P. Cerný, and S. Zdancewic. Preserving secrecy under refinement. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Proc. 33nd International Colloquium on Automata, Languages and Programming (ICALP)*, number 4052 in Lecture Notes in Computer Science, pages 107–118, 2006.

[6] G. Bella and E. Riccobene. Formal analysis of the kerberos authentication system. *Journal of Universal Computer Science*, 3(12):1337–1381, 1997.

[7] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 419–428, 1998.

[8] K. Bhargavan, R. Corin, P.-M. Deniélou, C. Fournet, and J. J. Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *Proc. 22th IEEE Computer Security Foundations Symposium (CSF)*, pages 124–140, 2009.

[9] P. Bieber and N. Boulahia-Cuppens. Formal development of authentication protocols. In *Sixth BCS-FACS Refinement Workshop*, 1994.

[10] P. Bieber, N. Boulahia-Cuppens, T. Lehmann, and E. van Wickeren. Abstract machines for communication security. In *Proc. 6th IEEE Computer Security Foundations Workshop (CSFW)*, pages 137–146, 1993.

[11] C. Boyd. Security architectures using formal methods. *IEEE Journal on Selected Areas in Communications*, 11(5), 1993.

[12] M. J. Butler. On the use of data refinement in the development of secure communications systems. *Formal Aspects of Computing*, 14(1):2–34, 2002.

[13] I. Cervesato, C. Meadows, and D. Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations*, pages 48–61, Washington, DC, USA, 2005. IEEE Computer Society.

[14] V. Cortier, M. Rusinowitch, and E. Zalinescu. Relating two standard notions of secrecy. *Logical Methods in Computer Science*, 3(3), 2007.

[15] V. Cortier, B. Warinschi, and E. Zalinescu. Synthesizing secure protocols. In *Proc. 12th European Symposium on Research in Computer Security (ESORICS)*, volume 4734 of *Lecture Notes in Computer Science*, pages 406–421. Springer, 2007.

[16] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositionl logic for security protocols. *Journal of Computer Security*, 13:423–482, 2005.

[17] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[18] ISO. Information Technology – Security Techniques – Entity Authentication Mechanisms – Part 3: Entity Authentication Using a Public-key Algorithm ISO/IEC 9798-3. International Standard, 2nd edition, 1998.

[19] J. Jürjens. Secrecy-preserving refinement. In *Proc. 10th Symposium on Formal Methods Europe (FME 2001)*, number 2021 in Lecture Notes in Computer Science, pages 135–152. Springer, 2001.

[20] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. *Journal of Cryptology*, 20(1):85–113, 2007.

[21] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software — Concepts and Tools*, 17:93–102, 1996.

[22] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, 1997.

[23] N. A. Lynch. I/O automaton models and proofs for shared-key communication systems. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, pages 14–29, 1999.

[24] H. Mantel. Preserving information flow properties under refinement. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 78–91, 2001.

[25] U. M. Maurer and P. E. Schmid. A calculus for secure channel establishment in open networks. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS)*, pages 175–192, 1994.

[26] A. McIver and C. C. Morgan. Sums and lovers: Case studies in security, compositionality and refinement. In *Formal Methods (FM 2009)*, pages 289–304, 2009.

[27] S. Mödersheim and L. Viganò. Secure pseudonymous channels. In *Proc. 14th European Symposium on Research in Computer Security (ESORICS)*, volume 5789 of *Lecture Notes in Computer Science*, pages 337–354. Springer, 2009.

[28] C. Morgan. The shadow knows: Refinement of ignorance in sequential programs. In *Mathematics of Program Construction (MPC 2006)*, volume 4014 of *LNCS*, pages 359–378, 2006.

[29] R. Needham and M. D. Schroeder. Using encryption for authentication in large data networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[30] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[31] L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[32] D. Pavlovic and C. Meadows. Deriving secrecy in key establishment protocols. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*, pages 384–403, 2006.