Refining Authenticated Key Agreement with Strong Adversaries

Joseph Lallemand
ENS Cachan, Université Paris-Saclay, and
Loria, Inria Nancy-Grand Est, France
Email: joseph.lallemand@loria.fr

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

***Abstract*—We develop a family of key agreement protocols that are correct by construction. Our work substantially extends prior work on developing security protocols by refinement. First, we strengthen the adversary by allowing him to compromise different resources of protocol participants, such as their long-term keys or their session keys. This enables the systematic development of protocols that ensure strong properties such as perfect-forward secrecy. Second, we broaden the class of protocols supported to include those with non-atomic keys and equationally defined cryptographic operators. We use these extensions to develop key agreement protocols including signed Diffie-Hellman and the core of IKEv1 and SKEME.**

1. Introduction

Key agreement protocols are a central element of many security architectures and hence methods for their rigorous development and security analysis are of vital importance. In this paper we propose an approach to developing key agreement protocols that are correct by construction in the presence of strong adversaries.

Our work is motivated by, and brings together, two separate lines of research. The first is on models and verification methods for security protocols that are secure against strong, but realistic, adversaries. The cryptographic community has developed numerous adversary models for evaluating the security of key agreement protocols, *e.g.*, [1]–[7]. These models formalize different kinds of adversarial capabilities. For example, in addition to the adversary being active and in control of all network traffic (sometimes called the "Dolev-Yao model"), the adversary can also compromise different resources of the protocol participants, such as their long term secrets, their session keys, parts of their session states, and even their random number generators. This makes it possible to distinguish between the protocols that provide security guarantees when some of these resources are compromised and those that do not. To enable automatic proofs, symbolic versions of these adversary models have been formalized and incorporated into security protocol model checkers, *cf.* [8], [9].

In contrast to the first line of research, which focuses on the post-hoc analysis of existing designs, the second line is on developing protocols that are correct by construction. Refinement is a popular formalism for developing systems together with their correctness guarantees and it has been applied to construct systems of all kinds [10]–[12]. Our starting point here is our previous work [13], [14], where we proposed a four-level refinement strategy for security protocols. We presented abstractions and an associated development method where protocols are designed through a sequence of refinement steps, while preserving correctness guarantees at each step. We formalized a supporting infrastructure in the theorem prover Isabelle/HOL [15] and used it to develop basic authentication and key establishment protocols.

We now briefly review our four-level refinement strategy and the related abstractions. Level 0, the most abstract level, provides simple protocol-independent transition system models of security properties, namely, secrecy and authentication. On Level 1, we model the protocol's roles and runs, but instead of exchanging messages on a communication medium, the participants communicate by reading each others' local store. On Level 2, the protocol participants exchange messages on communication channels with security properties, *e.g.*, insecure, authentic, confidential, and secure channels. We also model an adversary whose access to these channels is restricted depending on the channels' properties. Finally, on Level 3, channels are implemented by cryptographic operations and the adversary is a standard Dolev-Yao adversary.

Approach taken. In this paper, we combine the two lines of research described above: we present a method for developing security protocols using stepwise refinement that are correct in the presence of strong adversaries. We build on and extend the above refinement strategy to bridge symbolic models at different abstraction levels and we use this to systematically derive a family of Diffie-Hellman-based key agreement protocols, including signed Diffie-Hellman and the core of particular modes of IKEv1 and SKEME. We have formalized the entire approach, including the relevant infrastructure, our case study, and all proofs in Isabelle/HOL.

While our work in [13], [14] served as a good starting point, it had severe limitations. First, it cannot be used to model key agreement protocols where each party contributes information to the key, since the messages on the first three levels of abstraction, including the secrets to be protected,

were required to be atomic. Second, it could not model equational theories such as Diffie-Hellman exponentiation. This rules out advanced key agreement protocols, such as those based on Diffie-Hellman. Third, it only supported a limited adversary model: the standard Dolev-Yao model with the static compromise of honest agents' long-term keys. Hence, one cannot use it to prove stronger properties such as perfect forward secrecy and key independence, which are desirable for key agreement protocols. Finally, the implementation of channels with security properties (*e.g.*, authentic or confidential channels) as cryptographic messages was done separately for each protocol, thereby preventing their reuse. Overcoming these limitations required a major redesign for the results described here.

Contributions. Our main contributions are to substantially extend the scope of previous work and to apply the resulting framework to a non-trivial case study. This includes both extending the class of protocols supported and enabling the verification of secrecy and authentication properties with respect to strong adversaries. This has several aspects.

First, this required a major redesign of the framework of [13], [14]. This includes the modeling of non-atomic messages on all levels, adding support for equational reasoning by quotienting the set of messages by an equational theory associated with Diffie-Hellman exponentiation, and incorporating notions from cryptographic security definitions such as partner and test runs.

Second, following [8], [9], we have strengthened the adversary's capabilities by enabling him to dynamically compromise different resources of the protocol participants. Namely, we add two forms of agent compromise and session key compromise. The resulting symbolic adversary models correspond to Bellare et al.'s computational models [2], [3], [5]. By establishing secrecy properties in the presence of an adversary endowed with these capabilities, the derived key agreement protocols satisfy stronger guarantees, namely, perfect forward secrecy and key independence.

Third, we implement channels in a parametric way, under a set of assumptions that any concrete implementation must satisfy. This approach sheds light on the conditions that are needed for such implementations to be secure. It also makes our approach more modular in that we can derive a parametric Level 3 implementation that can be instantiated in many different ways, for example, using symmetric or asymmetric cryptography or some combination. We provide two such implementations, based respectively on symmetric and asymmetric cryptography, and use these to instantiate parametric Level 3 protocols into concrete ones.

Finally, we validate our approach by developing a family of Diffie-Hellman-based key agreement protocols. This shows how one can use our refinement framework to develop protocols that are secure against strong forms of adversarial compromise, analogous to those found in cryptographic security definitions.

All results reported in this paper have been verified using the Isabelle/HOL theorem prover. Full proof scripts, including all definitions and theorems are available online [16].

Structure of paper. The remainder of this paper is organized as follows. In Section 2 we introduce background on Isabelle/HOL and refinement. In Section 3 we provide a more detailed, but still informal, overview of our development approach. In Section 4 we present our development of a family of key agreement protocols satisfying strong security properties. In Section 5 we make further discuss related work and we draw conclusions in Section 6.

2. Background

We introduce background on Isabelle/HOL and refinement, following the presentation in [14].

2.1. Isabelle/HOL and notation

Isabelle is a generic, tactic-based theorem prover. We have used Isabelle's implementation of higher-order logic, Isabelle/HOL [15], for our development. To enhance readability, we will use standard mathematical notation where possible.

We use $\equiv$ for definitional equalities. The notation $f : A \rightharpoonup B$ denotes a partial function (or map) f from A to B with domain $dom(f)$. To specify a concrete map, we write $f = \{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$. Here, we have $dom(f) = \{a_1, \ldots, a_n\}$. For a function or binary relation $R \subseteq A \times B$ and set $X \subseteq A$, we define the image of X under R by $R(X) \equiv \{y \in B \mid \exists x \in X.\, (x, y) \in R\}$. We use inductively defined datatypes. For example, the type of lists is defined by $list(A) \equiv \mathsf{Nil} \mid \mathsf{Cons}(A, list(A))$. We define multisets over A by $multiset(A) \equiv A \to \mathbb{N}$. For $X \in multiset(A)$, the term $X(e)$ denotes the multiplicity of e in X and we define multiset union by $(X \uplus Y)(e) = X(e) + Y(e)$. Record types may be defined, *e.g.*, $point \equiv (\!|\, x \in \mathbb{N},\ y \in \mathbb{N} \,|\!)$ with elements like $r \equiv (\!|\, x = 1, y = 2 \,|\!)$ and projections $r.x$ and $r.y$. When it is clear from the context, we sometimes omit the field names, *e.g.*, $r = (\!|\, 1, 2 \,|\!)$. The term $r(\!|\, x := 3 \,|\!)$ denotes r, where x is updated to 3, *i.e.*, $(\!|\, x = 3, y = 2 \,|\!)$. The type $cpoint \equiv point + (\!|\, c \in color \,|\!)$ extends $point$ with a $color$ field. For record types T and U, where T includes the set F of fields of U, we denote the projection from T to U by π_F: $T \to U$. For example, $\pi_{\{x,y\}} : cpoint \to point$ with $\pi_{\{x,y\}}((\!|\, 1, 2, red \,|\!)) = (\!|\, 1, 2 \,|\!)$.

2.2. Refinement

A development by refinement starts from a set of system requirements and environment assumptions. We then construct a series of models resulting in a system that fulfills the requirements provided it runs in an environment satisfying the assumptions. We summarize our refinement theory that we developed in Isabelle/HOL, which is inspired by [11], [12].

Our models are *transition systems* $T = (\Sigma, \Sigma_0, \to)$. The state space Σ is a record type, whose fields are the state variables, $\Sigma_0 \subseteq \Sigma$ are the initial states, and the transition relation $\to$ is a finite union of parametrized relations, called *events*. Events have the form

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

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

Example 2.1. Consider an abstract file transfer protocol specification $T_f \equiv (\Sigma_f, \Sigma_f, \rightarrow_f)$, where $\Sigma_f \equiv (\!| f \in \mathit{file} |\!)$ with $\mathit{file} \equiv I \rightarrow D$ for a finite index set I and a set of data blocks D and $\rightarrow_f \equiv \mathit{xfer}_f$. The event $\mathit{xfer}_f \equiv \{(s, s') \mid s'.f := g\}$ transfers a given file g in one shot to f. All states are possible initial states.

In this paper, our notion of refinement is functional simulation [11], although our Isabelle/HOL development actually supports more general refinements. We say T_c *refines* T_a using the refinement mapping $\pi : \Sigma_c \rightarrow \Sigma_a$, written $T_c \sqsubseteq_\pi T_a$, if the following two conditions are met. (1) Each concrete initial state maps to some abstract initial state, *i.e.*, $\pi(\Sigma_c^0) \subseteq \Sigma_a^0$. (2) For each concrete event $evt_c(\vec{x})$ (with guards G_c, state variables $\vec{v}$, and update functions $\vec{f_c}$) we identify an abstract event $evt_a(\vec{z})$ (with guards G_a, state variables $\vec{u}$, and update functions $\vec{f_a}$) simulating it, *i.e.*, $\pi^{-1}; evt_c(\vec{x}); \pi \subseteq evt_a(\vec{w}(\vec{x}))$, where ';' is relational composition and the functions $\vec{w}(\vec{x})$ reconstruct the parameters for evt_a from those of evt_c. This condition decomposes into two proof obligations, called *guard strengthening* and *action refinement*, both under the premises $s \in \Sigma_c$ and $G_c(\vec{x}, s)$:

(GRD) $G_a(\vec{w}(\vec{x}), \pi(s))$

(ACT) $\pi(s)(\!| \vec{u} := \vec{f_a}(\vec{w}(\vec{x}), \pi(s)) |\!) = \pi(s(\!| \vec{v} := \vec{f_c}(\vec{x}, s) |\!))$

Guard strengthening requires that if the concrete event is enabled then so is the abstract one. Action refinement expresses that the two states resulting from the execution of the abstract and concrete actions are again related by π. We assume that all models include a special event $skip$ (the identity relation), which new concrete events must refine.

We express the desired properties of a system T as invariants. These are supersets of T's set of reachable states, $reach(T)$. We can use an invariant J of T_c to soundly strengthen the premise $s \in \Sigma_c$ of the refinement proof obligations above to $s \in J$. In this case, we may annotate the refinement with the invariant: $T_c \sqsubseteq_\pi^J T_a$.

The following result implies that invariants are preserved by a series of subsequent refinements.

Proposition 2.1. *Suppose* $T_2 \sqsubseteq_\pi T_1$. *Then*
1) $T_3 \sqsubseteq_{\pi'} T_2$ *implies* $T_3 \sqsubseteq_{\pi \circ \pi'} T_1$, *and*
2) $reach(T_1) \subseteq J$ *implies* $\pi(reach(T_2)) \subseteq J$.

Example 2.2. We define a "protocol" implementing the file transfer system T_f by $T_p \equiv (\Sigma_p, \Sigma_p^0, \rightarrow_p)$, where $\Sigma_p \equiv \Sigma_f + (\!| b \in I \rightharpoonup D |\!)$ extends the state Σ_f with a buffer b. The set $\Sigma_{p,0}$ consists of initial states of the form $(\!| f = f_0, b = \emptyset |\!)$ for some $f_0 \in \mathit{file}$ and the empty buffer. The protocol nondeterministically transfers blocks of the file g into the buffer b from where it is assigned to f, once the transfer is complete. The transition relation $\rightarrow_p \equiv \mathit{xfer}_p \cup \bigcup_{i \in I} blk(i)$ is the union of two events:

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

and $\mathit{xfer}_p \equiv \{(s, s') \mid dom(b) = I \wedge s'.f := s.b\}$.

Let us establish a refinement between S_p and S_f, using the projection $\pi_f : \Sigma_p \rightarrow \Sigma_f$ as a refinement mapping (where we omit the singleton brackets around f). We focus on point (2), where we must show that $blk(i)$ refines $skip$ and that xfer_p refines xfer_f. The guard strengthening (GRD) proof obligation is trivial in both cases, since the abstract guards are true. The action refinement (ACT) proof obligation for $blk(i)$ and $skip$ (the identity relation) requires showing $(\pi_f(s(\!| b(i) := g(i) |\!)) = \pi_f(s)$, assuming $i \in I$. This holds trivially. In the action refinement for xfer_p and xfer_f, we must show that $\pi_f(s(\!| f := \pi_f(s).b |\!)) = \pi_f(s)(\!| f := g |\!)$ assuming $dom(b) = I$. To prove this, we need additional information about the relation between b and g, expressed as the invariant $I_p \equiv \{s \in \Sigma_p \mid \forall i \in dom(b).\ s.b(i) = g(i)\}$ of S_p. We first establish this invariant and then use it to prove $T_p \sqsubseteq_{\pi_f}^{I_p} T_f$.

In further refinements, one could develop a more realistic implementation, for example, by eliminating nondeterminism and by modeling a communication medium.

3. Overview of approach and development

In this section, we describe our refinement approach and how we develop key agreement protocols. We explain our development in a high-level way, focusing on the main ideas and design decisions. We also formally introduce some basic elements of our framework, but keep the discussion informal otherwise. In Section 4, we expand this description and present more formal details.

3.1. Protocol refinement framework

Our development framework is organized into a four-level refinement strategy. This strategy provides four views of security protocols, each at a different abstraction level. These levels are as follows.

Level 0 *Security property models.* These models provide simple, protocol-independent specifications of secrecy and authentication for which the requisite security invariants are trivial to establish.

Level 1 *Guard protocols.* We introduce protocol roles and runs. A run maintains the state of a protocol role instance during its execution. The runs communicate by reading each others' local stores, whereby logical conditions called *security guards* ensure secrecy or authenticity if required. The adversary only loosely interacts with the protocol: unguarded reads are completely non-deterministic and the adversary can learn any message that does not allow him to derive a protocol secret.

Level 2 *Channel protocols.* The runs communicate by exchanging messages over communication channels with

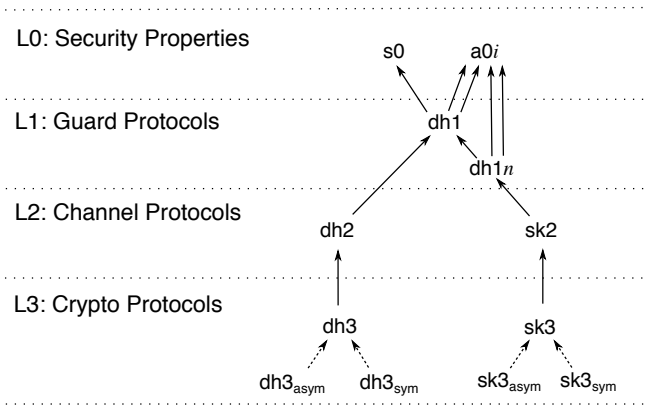

Figure 1. Refinement graph of our case study

security properties, such as authentic or confidential channels. The adversary interacts more closely with the protocol: he can access channels depending on their security properties and he can corrupt agents (and hence access their channels) as well as session keys.

Level 3 *Cryptographic protocols.* We refine the channel messages by cryptographic operations using long-term keys. This refinement is parametrized by the message implementation, subject to a set of assumptions. We can then obtain different protocol variants by instantiating the implementation in different ways and discharging the related assumptions. The adversary is a Dolev-Yao adversary with additional compromising capabilities.

We use the refinement framework from Section 2.2 to establish a protocol's secrecy and authentication properties by refinement of the corresponding Level 0 models. We do this at the highest possible abstraction level, namely Level 1. The crucial point is that these properties, once proved as invariants at Level 0, are inherited by subsequent refinements down to the final models at Level 3 (see Proposition 2.1).

3.2. Methodology and key agreement case study

We start each development from a set of requirements and environment assumptions. The latter include the adversarial capabilities and the cryptographic setup. The goal of two-party authenticated key agreement is to establish:

P1. a secret session key shared by the two parties, and
P2. agreement on (at least) each other's role and identity and on the session key.

The network environment is characterized by a Dolev-Yao adversary with additional compromising capabilities. Moreover, we consider both symmetric and asymmetric cryptographic setups.

Figure 1 gives an overview of our development of a family of key agreement protocols. The solid arrows represent refinements and the dashed arrows instantiations. Our development method proceeds as follows:

L0. We use the Level 0 models to formalize the security properties P1 and P2 above.

L1. We determine the messages to be exchanged (the DH public keys) or computed (DH key) and their individual security properties, which we enforce by appropriate security guards. This results in the model dh1, from which all our other protocol models are derived. We establish its security properties by a refinement of the secrecy model (s0) and two refinements of the authentication model (a0i), one for each role. The model dh1n refines dh1 by adding nonces. Its direct refinements of a0i strengthen the authentication properties.

L2. We choose the channel types and messages such that receiving a message implies the corresponding security guards from Level 1 (guard strengthening), thus explicitly relating these messages to their expected security properties. We refine the models dh1 and dh1n differently: using authentic channels (dh2) and using confidential channels (sk2). The environment assumptions determine the adversary's capabilities.

L3. We refine the two channel protocol models into (parametric) cryptographic implementations (dh3 and sk3). We then instantiate these with specific channel implementations based on asymmetric and symmetric cryptography, as defined by our assumptions on the cryptographic setup. The model $\mathsf{dh3}_{\mathsf{asym}}$ represents a signature-based authenticated Diffie-Hellman protocol, while $\mathsf{sk3}_{\mathsf{asym}}$ models both the basic mode of the SKEME protocol [17] and the variant of IKEv1's aggressive mode using public-key encryption [18].

Note that, at each level, different design choices are possible.

We next introduce protocol messages and the Dolev-Yao adversary and then give a sketch of the models in Figure 1.

3.3. Messages and adversary derivation

We model protocol messages as terms constructed from a set of atomic messages and a signature of constructors. The atomic messages are agents A, B, natural number constants i, nonces N, tags, and long-term keys, *i.e.* public keys $\mathsf{pub}(A)$, private keys $\mathsf{pri}(A)$, and shared keys $\mathsf{shr}(A,B)$. The composed messages are pairing $\langle M_1, M_2\rangle$, hashing $\mathsf{h}(M)$, symmetric encryption $\{\!|M|\!\}_K$, message authentication codes (MACs) $[\![M]\!]_K$, asymmetric encryption $\{M\}_K$, signatures $[M]_K$, and exponentiation $\mathsf{exp}(X,Y)$, also written as X^Y.

We quotient these messages by the equivalence relation associated to the following single-equation equational theory:

$$\mathsf{exp}(\mathsf{exp}(x,y),z) = \mathsf{exp}(\mathsf{exp}(x,z),y)$$

This equation is needed for Diffie-Hellman key exchange protocols to be executable. A more detailed representation of the adversary capabilities requires additional algebraic structure in the exponent, *e.g.*, a group.

Dolev-Yao adversary. We model standard Dolev-Yao adversary capabilities using the closure operators $synth$ for message composition and $analz$ for decomposition; this is a variant of [19] that supports composed keys. Given a set of messages S, $synth(S)$ denotes the set of messages the

adversary can obtain by applying constructors to messages in S and $analz(S)$ is the set of messages he can extract by decomposing messages in S (*e.g.*, by projection or decryption). These operators are inductively defined using a set of rules. Here is an example rule:

$$\frac{\{\!|X|\!\}_K \in analz(S) \quad K \in synth(analz(S))}{X \in analz(S)} .$$

Note that $synth$ is required here for the derivation of composed keys K. We define the set of messages derivable by the adversary from those in S by $DY(S) \equiv synth(analz(S))$. We also use $parts(S)$ to denote the inductively defined set of all accessible subterms of terms in S, *i.e.*, the components of pairs and the cleartexts of encryptions and signatures. We describe the adversary's advanced compromising capabilities in Section 3.6 below.

We partly reuse Paulson's Isabelle/HOL message theory [19], but we extended it with composed keys and Diffie-Hellman exponentiation with the equational theory above.

Payload messages. At Level 3, we will implement channel messages with (symmetric or asymmetric) cryptographic operations using long-term keys and tags. We call these keys and the tags *implementation material* and we require that the protocol uses them only for this purpose. That is, the messages carried by the channels at Level 2 must not contain any implementation material. We call such messages *payload messages*. Our refinement proofs from Level 2 to Level 3 rely on the clear distinction between payload messages and valid implementations of channel messages (see Section 4.5). To achieve this, we use tags in the channel message implementations and we impose the restriction that protocols only accept and store payload messages in their frames (see Section 3.5). Alternatively, we could require that the protocol messages at Level 3 are fully decryptable [20] or well-typed [21], but we prefer our weaker condition here.

3.4. Level 0: Security properties

The protocol-independent models for secrecy and authentication are very simple. The secrecy model s0's state consists of two sets of messages: the adversary knowledge, ik, and the secrets, $secret$. There are two events: one to add a non-derivable message M (*i.e.*, $M \notin DY(ik)$) as a secret and one where the adversary learns any message that does not allow him to derive a secret. The secrecy invariant states $DY(ik) \cap secret = \emptyset$, *i.e.*, no secret is derivable.

We formulate authentication properties in terms of injective and non-injective agreement. Following standard practice [22], we formalize them using *signals* to mark well-defined stages in a role's execution. For an agreement of role A with role B on message M, (instances of) role A emit commit signals $\mathsf{Cmt}(A, B, M)$ to express the claim that there is a corresponding instance of role B that has previously emitted a running signal $\mathsf{Run}(A, B, M)$. We have two abstract models, which maintain a multiset of signals, $sigs$. The model a0n for non-injective agreement has the invariant that the existence of a commit signal implies the existence of a matching running signal. The model a0i for injective agreement guarantees the stronger invariant that there are not more commit signals than matching running signals. In each model, there is a running event, which adds a running signal to the multiset, and a commit event, which adds a commit signal guarded by a condition that maintains the respective invariant.

3.5. Level 1: Guard protocols

We introduce protocol runs, which represent executing instances of protocol roles. We model these as a function

$$runs \in rid \to (\!|role \in role, \\ owner \in agent, \\ partner \in agent, \\ frame \in var \rightharpoonup msg|\!)$$

from run identifiers in rid to records containing the role name (initiator or responder), the names of the owner and its intended partner, and the frame corresponding to the local message store. The frame maps variables to messages that are generated, computed, or received during the run's execution.

Next, we introduce three notions important for our development: partner runs, the a-priori guessing of runs, and the test run. Even though partnering is only used later at Level 2 to model session key compromise, we introduce it here to motivate some of our general design decisions.

Partner runs. We want to prove the secrecy of a given session key under a strong adversary who can compromise keys established in other sessions. To formalise session key compromise, we have to identify the partner runs, which are intended to share a session key. Intuitively, two runs are partners if they play complementary roles and agree on the participants and on the exchanged messages. To facilitate the formal definition of partnering, which we will give in Section 4.3, we guess the runs' final states including their frames in advance and use these for defining partnering.

Guessing. We guess the runs by fixing an arbitrary function $runs$ instead of making it a state variable. To track each run's progress, we use a state variable $prog \in rid \to \mathcal{P}(var)$, which maps each run identifier to the set of variables representing its frame's current domain. We then use guards in the protocol events to ensure that the actual protocol execution is consistent with the guessed runs.

We impose some conditions on $runs$. First, we store the nonces generated locally by each run R in its frame. These are of the fixed form $R\$l$, where l is a label. Together R and l ensure the nonce's uniqueness. This condition is without loss of generality and would be proved as an invariant in a setting with dynamically updated runs. Second, we require that the frames only store payload messages as discussed in Section 3.3.

Security properties and test run. Suppose we have a model where the adversary can unconditionally compromise certain entities such as private keys or session keys (as in computational models of authenticated key exchange [1]–[7]). In such a setting, security properties can then be expressed as invariants as follows:

$$\psi \equiv \forall R \in rid.\ clean(R) \longrightarrow \phi(R).$$

Here, $\phi(R)$ is the actual security property for run R, *e.g.*, "The adversary cannot deduce any secret of R". The predicate $clean(R)$ restricts the possible compromises to avoid an overly powerful adversary. For example, neither the private keys of R's owner or partner agents nor the session key of R or its partner runs should be compromised.

In our setting, security properties are proved as invariants of abstract (Level 0) models and then inherited by the protocol models at Levels 1-3 through refinement. These models inherit the variables $secret$, ik, and $sigs$ from Level 0. Since the notions of runs and compromise are unknown at Level 0, these invariants cannot refer to them. We now sketch in two transformation steps how the property ψ above is related to our abstract formalization of security properties at Level 0.

In the first step, we eliminate the quantification over all runs and the $clean$ predicate from the security property. We do this by expressing the security property ψ relative to an arbitrary but fixed run, called the *test run*. This notion is standard in computational models and was adapted to a symbolic model in [8]. Moreover, we model each of the adversary's compromising capabilities as a different event whose guards express the respective restrictions from $clean(test)$ (see Section 4.3). Hence, we obtain the property

$$\phi(test).$$

In the second step, we remove the remaining reference to the test run. We achieve this by restricting the use of the variables $secret$ and $sigs$ such that, in the resulting model, the properties are only checked for the (arbitrary) test run. For secrecy, only the test run can add secrets to $secret$. Secrecy then corresponds to our Level 0 model: the adversary cannot deduce any message in $secret$.

For authentication, we cannot restrict the signaling to the test run alone since its partner run must emit the complementary signal. However, if only these two runs could emit signals, agreement would hold trivially. Therefore, we restrict signal emissions to the test owner and its partner agent and prior to the test run's end. This ensures that signals are only emitted by honest (*i.e.*, uncompromised) agents, which avoids trivial property violations. We will formalize this as the predicate can_signal in Section 4.2. The resulting authentication property also matches our Level 0 models.

Our formulation of properties has several advantages. First, since the properties do not refer to runs or compromises, they are much simpler and more abstract. Second, this allows us to inherit Level 0 properties by refinement. Finally, our formulation is independent of the adversary's compromising abilities and enables a modular adversary model.

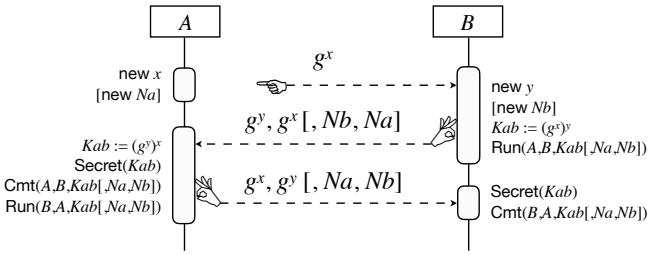

Figure 2. Key agreement guard protocols dh1 and dh1n (with nonces)

Security guards and adversary interaction. In order to authentically exchange and compare values, the runs read each others' frames. We achieve this using guards, called *authentication guards*, which access another run's frame to check the presence of certain values and read others. Likewise, *secrecy guards* ensure that messages that we would like to remain secret cannot be derived by the adversary.

At this level, there is only a loose interaction with the adversary: unprotected reads non-deterministically yield arbitrary values and the adversary can learn any message that does not help him derive a secret claimed by the protocol. This loose coupling between protocol and adversary simplifies security proofs at this abstraction level.

Guard protocols, with their memory-reading communication controlled by such *security guards*, provide an abstraction layer that enables a unified view on protocols that differ on lower levels. For example, all four key agreement protocols that we develop are derived from the same guard protocol. Other examples are challenge-response mechanisms using either authentic or confidential channels, which we can derive from a single guard protocol [13].

Guard protocol for key agreement. Figure 2 displays the information exchange between an initiator A and a responder B in the Level 1 models dh1 and dh1n, who share a public group generator g. In dh1, the initiator A generates a private key x and stores it alongside the public key g^x in her frame. Then the responder B generates a nonce y, computes g^y, and stores them in his frame. Then B non-authentically obtains g^x (indicated by the pointing hand), which may yield an arbitrary value instead. Next, A checks the presence of g^x in B's frame and reads g^y using an authentication guard (indicated by the picking hand). She then computes the shared secret as $Kab = (g^y)^x$. In the final step, B authentically reads g^x from A's frame, checks the presence of g^y, and computes Kab as $(g^x)^y$. In dh1n, nonces Na and Nb (bracketed in Figure 2) are also generated and exchanged.

3.6. Level 2: Channel protocols

Here, we introduce channels with security properties. We have four channel types: insecure, authentic, confidential, and secure. For informal use, we adopt the notation of [23]

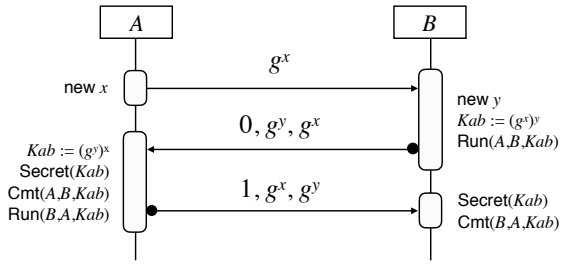

Figure 3. Key agreement protocol with authentic channels (dh2)

for channels where A sends a message M to B:

$$\begin{array}{ll} A \;\rightarrow\; B : M & \text{insecure channel} \\ A \bullet\!\!\rightarrow\; B : M & \text{authentic channel} \\ A \;\rightarrow\!\!\bullet\; B : M & \text{confidential channel} \\ A \bullet\!\!\rightarrow\!\!\bullet\; B : M & \text{secure channel} \end{array}$$

A bullet at a channel's endpoint indicates the respective agent's exclusive access to that end. For example, only A can write into an authentic channel $A \bullet\!\!\rightarrow B$, whereas only B can read from a confidential channel $A \rightarrow\!\!\bullet B$. A secure channel is both authentic and confidential. The adversary can read from non-confidential and write to non-authentic channels.

We also equip the adversary with various compromising capabilities. As explained in Section 3.5, we model different forms of compromise with respect to the test run. Moreover, the restrictions of some forms of compromise refer to the test run's partner runs. Here are the types of compromises that we model.

Compromise of other agents. The adversary can compromise any agent except the owner of the test run and its intended communication partner. These are excluded, since security properties are required to hold only between honest agents. This models a dynamically compromising Dolev-Yao adversary.

Compromise of owner. By allowing the adversary to compromise the test run's owner after the test run has finished, we can strengthen a standard secrecy property to *perfect forward secrecy*.

Session key compromise. The adversary can compromise session keys that belong neither to the test run nor to any of its partner runs. Showing secrecy in this setting additionally guarantees *key independence*.

Similar to [8], we model each adversary capability in a modular fashion by a single event. This enables us to consider arbitrary combinations of these capabilities. With the three events above, we cover the models by Bellare et al. [2], [3], [5]. We do not cover the models by Canetti and Krawczyk [4], [6], its extension in [7], or the model in [1], since these allow the adversary to reveal state, randomness, or the test agent's long-term keys. We will discuss the possibility of including these forms of compromise in Sections 5 and 6. For a more detailed discussion of the relationship to the computational models cited above, we refer the reader to [8].

Figure 3 shows the Level 2 model dh2 of our key agreement case study, which uses authentic channels to exchange the DH public keys. In the first message, A sends her public key to B. In the second and third messages, both public keys are sent on authentic channels from B to A and from A to B. We use the numbers 0 and 1 as tags to disambiguate the interpretation of messages. Figure 4 shows an alternative Level 2 model for key agreement, sk2, which uses confidential channels to exchange the nonces Na and Nb and insecure channels to exchange the DH public keys. The public keys are authenticated in the final two messages by MACing them using the key $K = \mathsf{h}(Na, Nb)$ derived from the nonces.

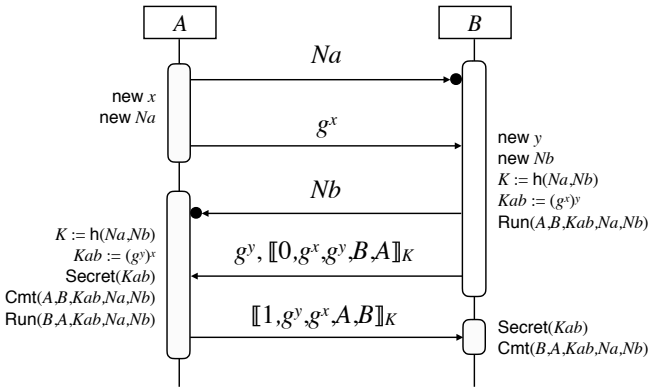

Figure 4. Key agreement with confidential channels (sk2)

Both of these Level 2 models refine the Level 1 model dh1. In the corresponding refinement proofs, one must show that the reception of an authentic message (in dh1) or a MAC (in dh1n) implies the authentication guard in the corresponding abstract event. This proof obligation, which we prove as an invariant, precisely captures in a logical form the property expected from the message.

3.7. Level 3: Cryptographic protocols

At this level, we refine the channel messages into cryptographic messages. We do this in a parametric way by stipulating the existence of a function $impl$ that implements the channel messages, without explicitly defining it. This function is required to satisfy a number of assumptions that characterize the different channel properties, that is, the adversary's capabilities to read from and write into channels on Level 2. The main conditions express what the adversary can learn from these messages and to which extent he can fake them. To define the refinement mapping, we partition the Level 3 adversary knowledge and relate different partitions to the channel messages and to the adversary knowledge at Level 2 (see Figure 6).

We obtain different instantiations of the channels by providing a concrete definition of the implementation function $impl$ using cryptographic operations with long-term keys to realize the different channels. We must also prove that the related assumptions hold for the given implementation. It suffices to do this once. One can thus build a library of different channel realizations that fit different cryptographic

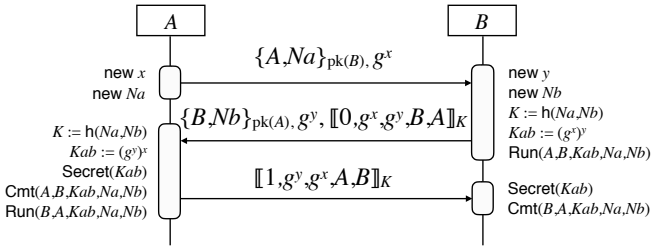

Figure 5. Key agreement protocol using asymmetric encryption ($\mathsf{sk3_{asym}}$)

setups. The channel implementations can then be reused in different contexts. In our development, we provide two such channel implementations, respectively based on:

asymmetric cryptography where we implement the channels using public-key encryption and digital signatures;

symmetric cryptography where we implement the channels using symmetric encryptions and MACs.

The Level 3 models dh3 and sk3 are parametrized by a channel implementation and so are the refinement mappings with the respective Level 2 models, dh2 and sk2. The instantiation of these Level 3 protocols with concrete channel implementations is thus performed *after* the refinement proof and incurs no further proof obligations. In our case study, we thus obtain symmetric and asymmetric realizations of the generic protocols dh3 and sk3 (see Figure 1) "for free". For instance, the model $\mathsf{sk3_{asym}}$, displayed in Figure 5, instantiates the generic model sk3 with the asymmetric channel implementation where confidential channels are realized by public-key encryption.

4. Development of key agreement protocols

We now present in more detail and formality how we use our refinement strategy to develop key agreement protocols. We focus our presentation on the SKEME protocol, depicted in the right branch of our refinement graph (Figure 1). Note that our development method is general and systematic at all levels and can be applied to other case studies.

4.1. Level 0: Security properties

At this level, we define protocol-independent models of secrecy and authentication, which we use to formalize security requirements.

Secrecy model. The state of this model consists of two sets of messages: the set of secrets $secret$ and the adversary knowledge ik. Both are initially empty.

This model has two events. The secret generation event $gen_{\mathsf{s0}}(M)$ marks a message M as a secret by adding it to $secret$, provided that it is not already deducible from ik:

$$\begin{aligned} gen_{\mathsf{s0}}(M) = \{(s, s') \mid \\ M \notin DY(s.ik) \wedge \\ s'.secret := s.secret \cup \{M\} \ \}. \end{aligned}$$

The adversary event $learn_{\mathsf{s0}}(M)$ represents the adversary learning M by adding it to ik, provided that it does not enable him to deduce a secret.

$$\begin{aligned} learn_{\mathsf{s0}}(M) = \{(s, s') \mid \\ s.secret \cap DY(s.ik \cup \{M\}) = \emptyset \wedge \\ s'.ik := s.ik \cup \{M\} \ \} \end{aligned}$$

The secrecy property is then expressed as an invariant stating that the adversary cannot derive any secret:

$$s.secret \cap DY(s.ik) = \emptyset.$$

Authentication models. We define two authentication models: $\mathsf{a0}n$ for *non-injective* and $\mathsf{a0}i$ for *injective* agreement. These models and the related properties are based on *signals*, which are emitted by the protocol events at lower levels. These are defined as follows:

$$\begin{aligned} signal \ \equiv \ & \mathsf{Cmt}(agent, agent, msg) \\ & \mid \mathsf{Run}(agent, agent, msg). \end{aligned}$$

The commit signal $\mathsf{Cmt}(A, B, M)$ represents the claim that agent A runs a session of the protocol with B, and expects that B is also running a session with A and that they agree on the data M. The complementary signal $\mathsf{Run}(A, B, M)$ simply represents the statement that agent B believes he is running a session of the protocol with agent A and knows some data M.

The state of both authentication models consists of a single field $sigs$, which is a multiset of signals:

$$\Sigma_{\mathsf{a0}(i,n)} \equiv (\!| \ sigs \in multiset(signal) \ |\!).$$

The agreement guarantee for A can be non-injective, *i.e.* A knows that B is running at least one session with data M, or injective, *i.e.* A knows that B is running at least as many sessions as her with this data. We formalize injective agreement as the following invariant. For all A, B, and M:

$$s.sigs(\mathsf{Cmt}(A, B, M)) \leq s.sigs(\mathsf{Run}(A, B, M)).$$

This means that there is an injection from $\mathsf{Cmt}(A, B, M)$ to $\mathsf{Run}(A, B, M)$ signals. In the non-injective case, we only require that the existence of a commit signal implies the existence of the matching running signal.

There are two events, one for emitting each type of signal. The $running(A, B, M)$ event adds a $\mathsf{Run}(A, B, M)$ signal to the multiset. It is identical in both $\mathsf{a0}n$ and $\mathsf{a0}i$.

$$\begin{aligned} running_{\mathsf{a0}(i,n)}(A, B, M) \equiv \{(s, s') \mid \\ s'.sigs := s.sigs \uplus \{\mathsf{Run}(A, B, M)\} \ \} \end{aligned}$$

The $commit(A, B, M)$ event adds a $\mathsf{Cmt}(A, B, M)$ signal to the multiset. This event differs in the two models. The version for injective agreement is

$$\begin{aligned} commit_{\mathsf{a0}i}(A, B, M) \equiv \{(s, s') \mid \\ s.sigs(\mathsf{Cmt}(A, B, M)) < s.sigs(\mathsf{Run}(A, B, M)) \wedge \\ s'.sigs := s.sigs \uplus \{\mathsf{Cmt}(A, B, M)\} \ \}. \end{aligned}$$

The guard ensures that the invariant is preserved. In the non-injective version of this event, the guard accordingly only requires the existence of a running signal.

4.2. Level 1: Guard protocols

At this level, we introduce the notion of protocol runs. As explained in Section 3.5, each run has a role, an owner agent, a partner agent, and a frame. The runs communicate by reading values directly from each other's memory without using an intermediate communication channel.

In our case study, we have two models at Level 1: dh1 for the basic exchange of the DH public keys and dh1n where nonces are added (Figure 2). Here, we will describe the model dh1n.

State and events. As explained in Section 3.5, the state of both Level 1 models is

$$\Sigma_{\mathsf{dh1}n} \equiv \Sigma_{\mathsf{s0}} + (\!| sigs_{\mathsf{Init}} \in multiset(signal), \\ sigs_{\mathsf{Resp}} \in multiset(signal), \\ prog \in rid \to \mathcal{P}(var) |\!).$$

It consists of (1) the message sets ik and $secret$, inherited from the secrecy model s0; (2) two multisets $sigs_{\mathsf{Init}}$ and $sigs_{\mathsf{Resp}}$, inherited from the two refinements of the agreement model a0i, one for showing that the initiator authenticates the responder and the other one vice versa; and (3) the mapping $prog$ from run identifiers to sets of variables representing each run's current domain. There are two kinds of events: the protocol events and the adversary's $learn$ event, which is identical to the $learn$ event of the model s0.

As explained in Section 3.5, we restrict signal emission to the test run's owner and its partner agent and it must occur before the test run's end. We define this condition as

$$can\_signal(s, A, B) \equiv \mathtt{end} \notin s.prog(test) \wedge \\ \{A, B\} = \{runs(test).owner, runs(test).partner\}.$$

Here, end is a frame variable that marks the end of a run's execution. We use typewriter font for frame variables.

As an example, we describe in detail the formal definition of the event for Step 3. This example illustrates the general form of a Level 1 protocol step.

$$step3_{\mathsf{dh1}n}(Ra, A, B, \sigma, GY, Nb) = \{(s, s') \mid$$
-- fix run Ra's role, agents, frame, and progress
$runs(Ra) = (\!|\, \mathsf{Init}, A, B, \sigma \,|\!) \wedge$
$s.prog(Ra) = \{\mathtt{nx}, \mathtt{gx}, \mathtt{na}\} \wedge$
-- check consistency with σ
$\sigma(\mathtt{gy}) = GY \wedge \sigma(\mathtt{nb}) = Nb \wedge \sigma(\mathtt{kab}) = GY^{\sigma(\mathtt{nx})} \wedge$
-- authentication guard
$authentication\_guard(s, Ra, A, B, \sigma, Nb, GY) \wedge$
-- secrecy guard
$(Ra = test \longrightarrow \sigma(\mathtt{kab}) \notin DY(s.ik)) \wedge$
-- update the run's progress, secrets, and signals
$s'.prog(Ra) := s.prog(Ra) \cup \{\mathtt{gy}, \mathtt{nb}, \mathtt{kab}, \mathtt{end}\}) \wedge$
$s'.secret := s.secret \cup \{\sigma(\mathtt{kab}) \mid Ra = test\} \wedge$
$s'.sigs_{\mathsf{Init}} := s.sigs_{\mathsf{Init}} \uplus$
$\{\mathsf{Cmt}(A, B, \langle \sigma(\mathtt{na}), Nb, \sigma(\mathtt{kab})\rangle) \mid$
$can\_signal(s, A, B)\} \wedge$
$s'.sigs_{\mathsf{Resp}} := s.sigs_{\mathsf{Resp}} \uplus \ldots \}$

The event's first guard fixes the role, Init, the names of the owner and partner agents, A and B, and the frame σ of the run Ra executing the step. The second guard checks that the run is ready to execute Step 3 by requiring that the current domain of its frame contains the variables bound in Step 1, where Ra generated the following terms:

$$\sigma = \{\mathtt{nx} \mapsto Ra\$\mathtt{nx},\ \mathtt{gx} \mapsto g^{Ra\$\mathtt{nx}},\ \mathtt{na} \mapsto Ra\$\mathtt{na}\}.$$

The next three guards bind the parameters GY and Nb to the variables gy and nb in the guessed frame σ and check that the variable kab indeed maps to the computed DH session key $GY^{\sigma(\mathtt{nx})}$. Next is the authentication guard:

$$authentication\_guard(s, Ra, A, B, \sigma, GY, Nb) \equiv \\ can\_signal(s, A, B) \longrightarrow (\exists Rb, \sigma'. \\ runs(Rb) = (\!|\, \mathsf{Resp}, B, A, \sigma' \,|\!) \wedge \\ \{\mathtt{ny}, \mathtt{gx}, \mathtt{gy}, \mathtt{na}, \mathtt{nb}, \mathtt{kab}\} \subseteq s.prog(Rb) \wedge \\ \sigma'(\mathtt{gx}) = \sigma(\mathtt{gx}) \wedge \sigma'(\mathtt{na}) = \sigma(\mathtt{na}) \wedge \quad \text{-- check} \\ \sigma'(\mathtt{gy}) = GY \wedge \sigma'(\mathtt{nb}) = Nb) \qquad \text{-- read}$$

Provided the can_signal condition holds, this guard ensures that there exists a responder run Rb owned by B with partner A that has progressed far enough and has the required values in its frame σ'. In particular, this guard authentically reads the values GY and Nb from σ' and ensures an agreement with Ra's frame σ on the variables gx and na. Note that this suffices to establish the required agreement on the session key kab, *i.e.*, property P2 from Section 3.2.

In case a message is read unauthentically, as for instance in Step 2, there is no authentication guard and the message that is read is completely unconstrained. This corresponds in lower levels to the fact that the message could have been faked by the adversary. However at this level of abstraction, the message that is actually read has no relation to the adversary or the adversary knowledge.

The final guard is the secrecy guard. This ensures for the test run that the adversary does not know the session key $\sigma(\mathtt{kab})$, thus safeguarding its declaration as a secret.

$$Ra = test \longrightarrow \sigma(\mathtt{kab}) \notin DY(s.ik)$$

Finally, we specify the event's actions. First, the run Ra's progress is updated to reflect the new messages learnt in this step. Note that the corresponding messages need not be recorded as they are already in the guessed frame σ. Since the authenticity of the other run's public key GY and the secrecy of the computed DH key $\sigma(\mathtt{kab}) = GY^{\sigma(\mathtt{nx})}$ are guaranteed by the security guards, the secret set is updated with $\sigma(\mathtt{kab})$, provided Ra is the test run. The signal multiset $sigs_{\mathsf{Init}}$ is updated with a commit signal for an agreement of the initiator A with the responder B on the nonces $Ra\$\mathtt{na}$ and Nb and on the session key $\sigma(\mathtt{kab})$, provided the $can\_signal(s, A, B)$

condition holds, and similarly for $sigs_{\mathsf{Resp}}$ (where we omit the added running signal). Note that can_signal is a condition for both these updates and the authentication guard and, similarly, the update of $secret$ and the secrecy guard are both conditioned on Ra being the test run.

By convention, our protocol events only use the variables $secret$ and $sigs$ as history variables, *i.e.*, to record messages or signals without referring to them in their guards. Hence, these variables do not influence the protocol's execution, which makes them suitable to express protocol properties. The variable $secret$ is however used in the guard of the adversary's $learn$ event at Level 1.

Refinements. As mentioned above, we have two models at Level 1, dh1 and dh1n, and several refinement relationships (see Figure 1). The model dh1n refines dh1 by adding nonces to the exchanged DH public keys. This refinement is easy to prove: the refinement mapping π_{11} simply removes the nonces from the dh1n frames to obtain the dh1 frames. The other fields remain the same.

Next, we show that dh1 refines s0. Then, by transitivity, dh1n also refines s0. The refinement with the Level 0 secrecy model is easily proved. The sets ik and $secret$ at Level 1 are identical to their counterparts in s0, whereas the other state variables, $prog$ and $sigs$, have no counterpart. Hence, the refinement mapping is defined as $\pi_{s01} \equiv \pi_{\{secret,ik\}}$. The protocol events that compute the session key K and mark it as a secret refine $gen_{\mathsf{s0}}(K)$, whereas the others refine the identity event $skip_{\mathsf{s0}}$. This is easy to show since the secrecy guard in the events that declare secrets at Level 1 directly corresponds to $gen_{\mathsf{s0}}(K)$'s guard in s0. The adversary event $learn$ is identical to its Level 0 counterpart and thus trivially refines it. The following result establishes the secrecy of the session key for each role, *i.e.*, property P1 from Section 3.2.

Proposition 4.1. dh1$n \sqsubseteq_{\pi_{11}}$ dh1 $\sqsubseteq_{\pi_{s01}}$ s0.

The refinement proofs of the agreement model a0i are slightly more involved. We establish two refinements between dh1 and a0i and two between dh1n and a0i. In dh1 the agreement is only on the DH key, while in dh1n it is extended to the two nonces. These stronger properties for dh1n cannot be inherited and require separate refinements of a0i.

These four refinement proofs (two for each model) are very similar. We focus here on the agreement of the initiator with the responder on the session key in dh1. In this case, only the $sigs_{\mathsf{Init}}$ multiset is kept, while the variables ik, $secret$, $sigs_{\mathsf{Resp}}$, and $prog$ disappear in the refinement. The $learn$ event, as well as every protocol event that does not emit any signal, refine $skip$. Suppose σ and σ' respectively are the initiator and responder's frame. The responder's $step2$ event emits the signal $\mathsf{Run}(A, B, \sigma'(\mathtt{kab}))$ and hence refines $running_{\mathsf{a0}i}(A, B, \sigma'(\mathtt{kab}))$, while the initiator's $step3$ emits $\mathsf{Cmt}(A, B, \sigma(\mathtt{kab}))$ and refines $commit_{\mathsf{a0}i}(A, B, \sigma(\mathtt{kab}))$. For the latter refinement, we must establish the corresponding guard strengthening proof obligation with the conclusion

$$sigs(\mathsf{Cmt}(A, B, \sigma(\mathtt{kab}))) < sigs(\mathsf{Run}(A, B, \sigma(\mathtt{kab}))).$$

We do this by proving invariants of the Level 1 model. One invariant states that the guarantees about the responder run Rb in the conclusion of the authentication guard implies that Rb has emitted its $\mathsf{Run}(A, B, \sigma'(\mathtt{kab}))$ signal. Another invariant implies that in the starting state of Step 3, there is no $\mathsf{Cmt}(A, B, \sigma(\mathtt{kab}))$ signal, while there is at least one matching running signal. This proves guard strengthening for Step 3. The following proposition establishes mutual injective agreement on the roles, agent identities, and the session key, *i.e.*, property P2 from Section 3.2.

Proposition 4.2. *Let* $I^{\mathsf{Init}}_{\mathsf{dh1}}$, $I^{\mathsf{Resp}}_{\mathsf{dh1}}$, $I^{\mathsf{Init}}_{\mathsf{dh1}n}$, *and* $I^{\mathsf{Resp}}_{\mathsf{dh1}n}$ *be the intersections of the invariants for each of the four refinements. Then, for all* $p \in \{\mathsf{dh1}, \mathsf{dh1}n\}$ *and* $r \in \{\mathsf{Init}, \mathsf{Resp}\}$, *we have*

$$p \sqsubseteq^{I^r_p}_{\pi_{sigs_r}} \mathsf{a0}i$$

In general, although we must prove the stated invariants for each protocol and agreement guarantee, their formulation is canonical and their proofs are similar.

4.3. Level 2: Channel protocols

We add communication channels between agents: rather than reading each other's memory, they now exchange messages over channels with intrinsic security properties.

Channel messages and basic adversary capabilities. We define a type $chan$ of *channel messages* representing messages on different types communication channels.

$$tag \equiv \mathsf{insec} \mid \mathsf{confid} \mid \mathsf{auth} \mid \mathsf{secure}$$
$$chan \equiv \mathsf{Chan}(tag, agent, agent, msg)$$

A channel message $\mathsf{Chan}(c, A, B, M)$ denotes a payload message M being sent from A to B on a channel of type c. We abbreviate $\mathsf{Chan}(\mathsf{insec}, A, B, M)$ by $\mathsf{Insec}(A, B, M)$ and likewise for $\mathsf{Confid}(A, B, M)$, $\mathsf{Auth}(A, B, M)$, and $\mathsf{Secure}(A, B, M)$. We now define the adversary's capabilities to eavesdrop payload messages on channels and to fake both payload and channel messages.

The adversary can *eavesdrop* payload messages on insecure and authentic (*i.e.*, non-confidential) channels as well as on channels where the agent at either endpoint is compromised. We formalize this capability as a closure operator, $extr(bad, ik, chan)$, denoting the set of extractable (payload) messages given a set bad of compromised agents, a set ik of payload messages representing the adversary knowledge, and a set $chan$ of channel messages. We define $extr$ using two rules. The first rule states that the adversary knowledge ik is included in the set of extractable messages and the second one is as follows:

$$\frac{\mathsf{Chan}(c, A, B, M) \in chan \qquad \begin{array}{l} c \in \{\mathsf{insec}, \mathsf{auth}\} \\ \vee A \in bad \vee B \in bad \end{array}}{M \in extr(bad, ik, chan)}$$

Next, we define the set of fakeable payload messages as the Dolev-Yao closure of the extractable messages.

$$dy\_fake\_msg(bad, ik, chan) \equiv DY(extr(bad, ik, chan))$$

Finally, we specify the set of fakeable channel messages. We define a closure operator $fake(bad, ik, chan)$ using two rules. One expresses that the set $chan$ is included in the set of fakeable messages and the other states that the adversary can fake insecure and confidential (*i.e.*, non-authentic) channel messages as well as messages on channels with a compromised agent at one endpoint. The latter rule is formalized as

$$\frac{M \in ik \quad c \in \{\mathsf{insec}, \mathsf{confid}\} \vee A \in bad \vee B \in bad}{\mathsf{Chan}(c, A, B, M) \in fake(bad, ik, chan)} .$$

This definition ignores that the message M could itself be faked. We therefore define the set of fakeable channel messages using dy_fake_msg as

$$dy\_fake\_chan(bad, ik, chan) \equiv \\ fake(bad, dy\_fake\_msg(bad, ik, chan), chan).$$

Note that if either of the agents A and B is compromised the adversary can eavesdrop and fake messages between these agents. This may appear strange at first, since in some cases it may result in a more powerful adversary than necessary, for instance, if the channels are later implemented by public-key cryptography. Indeed, the message $\{A, M\}_{\mathsf{pub}(B)}$ could implement $\mathsf{Confid}(A, B, M)$, but it cannot be decrypted if only A is compromised. However, this would not hold if M was symmetrically encrypted. We choose to model channels in a symmetric manner with respect to the two agents to enable refinements of the same Level 2 model by Level 3 models using either symmetric or asymmetric cryptography.

State and protocol events. The Level 2 state extends the Level 1 state with two variables: a variable $chan$ storing a set of channel messages and a variable bad holding the set of compromised agents.

$$\Sigma_{\mathsf{sk2}} \equiv \Sigma_{\mathsf{dh1}n} + (\!| \; bad \in \mathcal{P}(agent), chan \in \mathcal{P}(chan) \; |\!)$$

At this level, the runs communicate by writing and reading channel messages to and from the channel variable $chan$ instead of reading from each other's frame. We focus our discussion again on Step 3.

$$step3_{\mathsf{sk2}}(Ra, A, B, \sigma, GY, Nb) \equiv \{(s, s') \mid \\ \text{-- fix run } Ra\text{'s role, agents, frame, and progress} \\ runs(Ra) = (\!| \; \mathsf{Init}, A, B, \sigma \; |\!) \wedge \\ s.prog(Ra) = \{\mathtt{nx}, \mathtt{gx}, \mathtt{na}\} \wedge \\ \sigma(\mathtt{gy}) = GY \wedge \sigma(\mathtt{nb}) = Nb \wedge \sigma(\mathtt{kab}) = GY^{\sigma(\mathtt{nx})} \wedge \\ \text{-- receive channel messages} \\ \mathsf{Confid}(B, A, Nb) \in s.chan \wedge \\ \mathsf{Insec}(B, A, \langle GY, [\![0, \sigma(\mathtt{gx}), GY, B, A]\!]_{\mathsf{h}(\sigma(\mathtt{na}), Nb)}\rangle) \\ \quad \in s.chan \wedge \\ \text{-- actions:} \\ s'.chan := s.chan \cup \\ \quad \{\mathsf{Insec}(A, B, [\![1, GY, \sigma(\mathtt{gx}), A, B]\!]_{\mathsf{h}(\sigma(\mathtt{na}), Nb)})\} \wedge \\ \ldots \text{ (unchanged actions omitted) } \ldots \}$$

Here, the authentication and secrecy guards from the Level 1 model have been replaced by the reception of the confidential message containing Nb and the insecure message containing $\langle GY, [\![0, \sigma(\mathtt{gx}), GY, B, A]\!]_{\mathsf{h}(\sigma(\mathtt{na}), Nb)}\rangle$. We prove the associated guard strengthening as an invariant that we will discuss later in this section. This event sends the insecure message $\mathsf{Insec}(A, B, [\![1, GY, \sigma(\mathtt{gx}), A, B]\!]_{\mathsf{h}(\sigma(\mathtt{na}), Nb)})$ as a response. We elide the remaining actions since they are identical to those in the model $\mathtt{dh1}n$.

Adversary events. At this level, we explicitly model the adversary's capabilities to fake messages and to compromise different entities. First, he can fake both payload and channel messages. We therefore define two events using the closure operators defined at the beginning of this section.

$$dy\_fake\_msg_{\mathsf{sk2}}(M) \equiv \{(s, s') \mid \\ M \in dy\_fake\_msg(s.bad, s.ik, s.chan) \wedge \\ s'.ik := s.ik \cup \{M\} \}$$

$$dy\_fake\_chan_{\mathsf{sk2}}(m) \equiv \{(s, s') \mid \\ m \in dy\_fake\_chan(s.bad, s.ik, s.chan) \wedge \\ s'.chan := s.chan \cup \{m\} \}$$

Second, the adversary can compromise agents and session keys. The following event models the compromise of an agent other than the test run's owner or its partner. This models Dolev-Yao-style (dynamic) agent compromise and allows the adversary to participate in protocol runs by impersonating any of the compromised agents.

$$lkr\_others_{\mathsf{sk2}}(A) \equiv \{(s, s') \mid \\ A \notin \{runs(test).owner, runs(test).partner\} \wedge \\ s'.bad := s.bad \cup \{A\} \}$$

We also model two adversary events that go beyond a standard Dolev-Yao model. The following event models the compromise of any agent, including the test run's owner or partner, after the test run has ended. The presence of this adversary event strengthens secrecy to perfect forward secrecy.

$$lkr\_after_{\mathsf{sk2}}(A) \equiv \{(s, s') \mid \\ \mathtt{end} \in s.prog(test) \wedge \quad \text{-- test run ended} \\ s'.bad := s.bad \cup \{A\} \}$$

The final adversary event allows the adversary to compromise a run and steal its session key. To avoid trivially breaking the secrecy of the test run's session key, we must exclude the test run as well as any potential partner runs (with whom it may share this key) from this compromise.

$$skr_{\mathsf{sk2}}(R, K) \equiv \{(s, s') \mid \\ R \neq test \wedge R \notin partner\_runs(test) \wedge \\ \mathtt{kab} \in s.prog(R) \wedge runs(R).frame(\mathtt{kab}) = K \wedge \\ s'.ik := s.ik \cup \{K\} \}$$

Showing secrecy in the presence of this event establishes key independence.

Level 2	*Level 1*
protocol event (sk2)	protocol event (dh1n)
$dy\_fake\_msg_{\mathsf{sk2}}(M)$	$learn_{\mathsf{dh1}n}(M)$
$dy\_fake\_chan_{\mathsf{sk2}}(m)$	$skip_{\mathsf{dh1}n}$
$lkr\_others_{\mathsf{sk2}}(A)$	$skip_{\mathsf{dh1}n}$
$lkr\_after_{\mathsf{sk2}}(A)$	$skip_{\mathsf{dh1}n}$
$skr_{\mathsf{sk2}}(R, K)$	$learn_{\mathsf{dh1}n}(K)$

Table 1. REFINEMENT BETWEEN THE LEVEL 1 AND 2 EVENTS

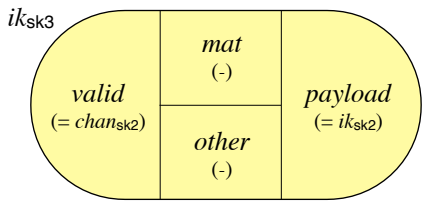

Figure 6. Partition of L3 adversary knowledge and refinement of L2 state

What remains is to precisely define the notion of partnering. Following the discussion in Section 3.6, we define the set of partner runs of a run R as follows:

$$
\begin{aligned}
&matching(\sigma, \sigma') \equiv \\
&\quad \forall x \in dom(\sigma) \cap dom(\sigma').\, \sigma(x) = \sigma'(x) \\
&partner\_runs(R) \equiv \{R' \mid \\
&\quad runs(R).role = compl\_role(runs(R').role) \wedge \\
&\quad runs(R).owner = runs(R').partner \wedge \\
&\quad runs(R).partner = runs(R').owner \wedge \\
&\quad matching(runs(R).frame, runs(R').frame)\ \}
\end{aligned}
$$

The run R and a partner run R' must play complementary roles (function $compl\_role$) and agree on their respective partners. Furthermore, the frames of R and R' must match on common variables, *i.e.*, those in the domain of both frames.

Invariants and refinement. The refinement mapping between the Level 2 and Level 1 state is $\pi_{12} \equiv \pi_F$, where F is the set of all fields of dh1n. Table 1 describes how events are refined. Each protocol event refines its Level 1 counterpart. The $dy\_fake\_msg_{\mathsf{sk2}}$ and skr_{sk2} events add messages to ik_{sk2}, and therefore refine $learn_{\mathsf{dh1}n}$. All other adversary events only change the channel ($dy\_fake\_chan_{\mathsf{sk2}}$) or the bad_{sk2} set (long-term key reveal events), which both have no Level 1 counterpart. Hence, these events refine $skip_{\mathsf{dh1}n}$.

We have proved several invariants for the model sk2. Two of these are needed to discharge the authentication guard strengthening proof obligations in the refinement proofs of Steps 3 and 4. For Step 3, this proof obligation is

$$
\begin{aligned}
&runs(Ra) = (\!|\, \mathsf{Init}, A, B, \sigma \,|\!) \wedge \\
&\ldots(\text{other guards of } step3_{\mathsf{sk2}} \text{ omitted})\ldots \\
&\mathsf{Insec}(B, A, \langle GY, [\![0, \sigma(\mathtt{gx}), GY, B, A]\!]_{\mathsf{h}(\sigma(\mathtt{na}),Nb)}\rangle) \\
&\quad \in s.chan \wedge \\
&can\_signal(s, A, B) \longrightarrow \\
&\quad (\exists Rb, \sigma'.\ runs(Rb) = (\!|\mathsf{Resp}, B, A, \sigma'|\!) \wedge \\
&\qquad \{\mathtt{ny}, \mathtt{gx}, \mathtt{gy}, \mathtt{na}, \mathtt{nb}, \mathtt{kab}\} \subseteq s.prog(Rb) \wedge \\
&\qquad \sigma'(\mathtt{gx}) = \sigma(\mathtt{gx}) \wedge \sigma'(\mathtt{na}) = \sigma(\mathtt{na}) \wedge \\
&\qquad \sigma'(\mathtt{gy}) = GY \wedge \sigma'(\mathtt{nb}) = Nb),
\end{aligned}
$$

where all variables are implicitly universally quantified.

This implication is implied by the following invariant:

$$
\begin{aligned}
&inv4_{\mathsf{sk2}} \equiv \{s. \mid \forall Ra, A, B, \sigma, GY, Nb. \\
&\quad runs(Ra) = (\!|\, \mathsf{Init}, A, B, \sigma \,|\!) \wedge \{A, B\} \cap s.bad = \emptyset \wedge \\
&\quad [\![0, \sigma(\mathtt{gx}), GY, B, A]\!]_{\mathsf{h}(\sigma(\mathtt{na}),Nb)} \\
&\qquad \in parts(extr(s.bad, s.ik, s.chan)) \longrightarrow \\
&\qquad (\exists Rb, \sigma'.\ runs(Rb) = (\!|\mathsf{Resp}, B, A, \sigma'|\!) \wedge \\
&\qquad\quad \{\mathtt{ny}, \mathtt{gx}, \mathtt{gy}, \mathtt{na}, \mathtt{nb}, \mathtt{kab}\} \subseteq s.prog(Rb) \wedge \\
&\qquad\quad \sigma'(\mathtt{gx}) = \sigma(\mathtt{gx}) \wedge \sigma'(\mathtt{na}) = \sigma(\mathtt{na}) \wedge \\
&\qquad\quad \sigma'(\mathtt{gy}) = GY \wedge \sigma'(\mathtt{nb}) = Nb)\ \}.
\end{aligned}
$$

In this invariant, we have weakened two premises of the guard strengthening proof obligation to enable its inductive proof. Namely, the premise $[\![0, \sigma(\mathtt{gx}), GY, B, A]\!]_{\mathsf{h}(\sigma(\mathtt{na}),Nb)} \in parts(extr(s.bad, s.ik, s.chan))$ replaces the premise $\mathsf{Insec}(B, A, \langle GY, [\![0, \sigma(\mathtt{gx}), GY, B, A]\!]_{\mathsf{h}(\sigma(\mathtt{na}),Nb)}\rangle) \in s.chan$ and the premise $\{A, B\} \cap bad = \emptyset$ replaces the condition $can\_signal(s, A, B)$. We establish that $can\_signal(s, A, B)$ implies $\{A, B\} \cap bad = \emptyset$ as an auxiliary invariant.

Proposition 4.3. *Let* I_{sk2} *be the intersection of the invariants of* sk2. *Then* $\mathsf{sk2} \sqsubseteq^{I_{\mathsf{sk2}}}_{\pi_{12}} \mathsf{dh1}n$.

4.4. Parametric channel implementation

We parametrize our Level 3 models with an implementation function $impl\colon chan \to msg$ mapping channel messages to cryptographic messages. We formulate a set of assumptions on $impl$ that reflects the channel security properties that a concrete implementation must realize. The refinement of the Level 2 models will rely on these properties and thus holds for any compliant implementation function.

To aid subsequent refinement proofs, we partition the set of messages as illustrated in Figure 6:

1) the implementation material $mat \equiv tag \cup ltk$ consisting of tags and long-term keys,
2) the set of payload messages,
$$payload \equiv \{M \mid \forall \text{subterms } M' \text{ of } M.\ M' \notin mat\},$$
3) the set of *valid implementations*,
$$valid \equiv \{impl(\mathsf{Chan}(c, A, B, M)) \mid M \in payload\},$$
4) and all other messages, $other$.

We call $core \equiv payload \cup valid \cup mat$ the core messages. The messages in $other$ can be seen as intermediate products arising in the construction of valid implementations: they contain implementation material but are not valid implementations themselves. We also define the following relation on agents:

$$broken(H) \equiv \{(A, B) \mid keys(A, B) \cap H \neq \emptyset\},$$

where $keys(A, B) = \{\mathsf{pri}(A), \mathsf{pri}(B), \mathsf{shr}(A, B), \mathsf{shr}(B, A)\}$ are the secret long-term keys for implementing channels between A and B.

We distinguish three types of assumptions. The basic assumptions separate the implementation messages from each other and from payload messages. The analyze and synthesize assumptions ensure that the Level 3 adversary is not stronger than the one at Level 2. Together they enable us to prove that the *other* messages (which have no Level 2 counterpart) are derivable from the core messages alone.

Basic assumptions. We assume the following basic properties of $impl$. For all $m, m' \in chan$ and $M \in msg$,

1) $impl$ is injective,
2) if $impl(m) \in valid$ and $impl(m') \in parts(impl(m))$ then $m' = m$,
3) $impl(m)$ is neither atomic nor in $payload$, and
4) if $M \in valid$ then $parts(M) \cap ltk = \emptyset$.

By the first two assumption, implementations of different channel messages are different and implementations must not be nested. The third property allows us to distinguish payload messages from implementations and the fourth avoids implementations that (potentially) expose long-term keys to the adversary.

Analyze assumptions. These assumptions bound what an adversary can extract from the implementation of channel messages. The first assumption states that a payload message extracted from a set of valid implementations can be derived from their payloads. For confidential messages, we have:

$$\begin{aligned} &G \subseteq payload \wedge H \subseteq core \longrightarrow \\ &\quad analz(impl(\mathsf{Confid}(agent \times agent, G)) \cup H) \\ &\quad \subseteq DY(G \cup H) \cup \overline{payload}, \end{aligned}$$

where $\mathsf{Confid}(Ag, G) \equiv \{\mathsf{Confid}(A, B, M) \mid (A, B) \in Ag \wedge M \in G\}$ for $Ag \subseteq agent \times agent$ and $G \subseteq msg$ and the $\overline{X}$ denotes the complement of the set X.

In addition, no payload message can be extracted from confidential or secure messages without knowing the relevant long-term keys.[1]

$$\begin{aligned} &G \subseteq payload \wedge H \subseteq core \wedge Ag \cap broken(H) = \emptyset \longrightarrow \\ &\quad analz(impl(\mathsf{Confid}(Ag, G)) \cup H) \subseteq DY(H) \cup \overline{payload} \end{aligned}$$

No such assumption is needed for non-confidential messages.

Synthesize assumptions. These assumptions express that if the adversary can construct the implementation I of a channel message with payload M from a set H of core messages then, unless I is already in H, he must also be able to construct M and know the required long-term keys of A and B. For example, for secure messages:

$$\begin{aligned} &impl(\mathsf{Secure}(A, B, M)) \in DY(H) \wedge H \subseteq core \longrightarrow \\ &\quad impl(\mathsf{Secure}(A, B, M)) \in H \vee \\ &\quad (M \in DY(H) \wedge (A, B) \in broken(H)). \end{aligned}$$

1. To simplify our presentation, this condition has a slightly stronger assumption than in our formalization.

The conjunct $(A, B) \in broken(H)$ is not needed for non-authentic messages.

Instantiation. We provide two concrete implementation functions: one using symmetric encryption and MACs, and one using asymmetric encryption and signatures. For example, we define the asymmetric implementation as follows:

$$\begin{aligned} impl_{\mathsf{asym}}(\mathsf{Insec}(A, B, M)) &= \langle \mathsf{insec}, A, B, M \rangle \\ impl_{\mathsf{asym}}(\mathsf{Confid}(A, B, M)) &= \{A, M\}_{\mathsf{pub}(B)} \\ impl_{\mathsf{asym}}(\mathsf{Auth}(A, B, M)) &= [B, M]_{\mathsf{pri}(A)} \\ impl_{\mathsf{asym}}(\mathsf{Secure}(A, B, M)) &= [\{\mathsf{secure}, A, M\}_{\mathsf{pub}(B)}]_{\mathsf{pri}(A)} \end{aligned}$$

Note that we use tags and agent identities in messages to fulfill the requirement that the implementation function is injective and that implementations cannot be nested.

We prove that the given implementations satisfy the assumptions. This must be done only once. In this way, one can build a library of implementations for different cryptographic setups. We can then instantiate any parametric Level 3 protocol with these implementations and establish, for free, that the protocol satisfies the desired security properties.

4.5. Level 3: Cryptographic protocols

At Level 3, we implement channel messages by cryptographic ones. Our Level 3 models and the refinement proofs are parametric in such an implementation and depend on a set of assumptions. We then define two different concrete implementations and show that they satisfy the assumptions. This allows us to instantiate the parametrized Level 3 protocols into concrete ones.

State and events. The Level 3 state is similar to the Level 2 state, but does not include the channel messages since, in the Dolev-Yao model, all communication goes through the adversary.

$$\Sigma_{\mathsf{sk3}} \equiv \Sigma_{\mathsf{dh1}n} + (\!| \; bad \in \mathcal{P}(agent) \; |\!).$$

We derive the Level 3 protocol events from those at Level 2 by simply replacing each channel message m by its implementation $impl(m)$ and references to $chan$ by ik.

We replace the adversary events dy_fake_msg and dy_fake_chan from Level 2 by the following standard Dolev-Yao adversary event:

$$\begin{aligned} &dy_{\mathsf{sk3}}(M) \equiv \{(s, s') \mid \\ &\quad M \in DY(s.ik) \wedge s'.ik := s.ik \cup \{M\} \}. \end{aligned}$$

The compromise events at Level 3 are similar to those on Level 2, but aside from adding the compromised agent to the set bad, they also add its long-term keys to the adversary knowledge ik.

Invariants and refinement. We show that the parametrized model $\mathsf{sk3}(impl)$ refines $\mathsf{sk2}$. Each protocol and compromise event refines its Level 2 counterpart. The situation is more involved for the Dolev-Yao event $dy_{\mathsf{sk3}}(M)$ and the variable ik_{sk3}. Since the Level 2 adversary does not know any

Level 3	*Level 2*
protocol event (sk3)	protocol event (sk2)
compromise event (sk3)	compromise event (sk2)
$dy_{\mathsf{sk3}}(M)$ if $M \in payload$	$dy\_fake\_msg_{\mathsf{sk2}}(M)$
$dy_{\mathsf{sk3}}(M)$ if $M \in valid$	$dy\_fake\_chan_{\mathsf{sk2}}(impl^{-1}(M))$
$dy_{\mathsf{sk3}}(M)$ if $M \in mat$	*skip* (M was already in ik_{sk3})
$dy_{\mathsf{sk3}}(M)$ if $M \in other$	*skip* ($other$ disappears at Level 2)

Table 2. REFINEMENT BETWEEN THE LEVEL 2 AND LEVEL 3 EVENTS

implementation material, we relate the payload messages in ik_{sk3} to those in ik_{sk2} and the valid implementations to the channel messages in $chan_{\mathsf{sk2}}$. This leads to the partitioning of ik_{sk3} and the refinement mapping π_{23} depicted in Figure 6. Formally, for a concrete state s, we have:

$$\begin{aligned}\pi_{23}(s).chan &= impl^{-1}(s.ik \cap valid)\\ \pi_{23}(s).ik &= s.ik \cap payload.\end{aligned}$$

All other fields shared with sk2 remain unchanged. The implementation material, mat, and the $other$ messages have no Level 2 counterpart. The event $dy_{\mathsf{sk3}}(M)$ then refines different Level 2 events, depending on which partition M is in (see Table 2).

We prove several invariants for this refinement of which we discuss the three main ones. The first main invariant relates the variable bad to the long-term keys in ik. It states that the adversary knows all public keys and at most the compromised agents' private and shared keys.

For the refinement to work, we must avoid strengthening the adversary compared to his Level 2 colleague. Hence, the second main invariant expresses that the messages in $other$ are deducible from the core messages in ik_{sk3}:

$$inv3_{\mathsf{sk3}} \equiv \{s \mid analz(s.ik) \subseteq DY(s.ik \cap core)\}.$$

Here, the restriction formulated in Section 3.3, that frames only store $payload$ messages, comes into play. This invariant holds only if the protocol events always send valid implementations. Since received messages may be faked by the adversary, non-payload elements may get into the frames and hence the sent messages. The restriction prevents this.

The final main invariant helps proving the guard strengthening obligation for the event $dy_{\mathsf{sk3}}(M)$ if $M \in payload$. It states that any $payload$ message derivable by the adversary is also in $dy\_fake\_msg(s.bad, s.ik \cap payload, impl^{-1}(s.ik))$.

While these invariants must be proved for every protocol, they are canonical and their proofs are very similar. Let I_{sk3} be the intersection of all these invariants.

Proposition 4.4. *Let H denote the assumptions on the implementation function described in Section 4.4. Let π_{23} be the refinement mapping described above. Then, for all $impl$ such that $H(impl)$,*

$$\mathsf{sk3}(impl) \sqsubseteq^{I_{\mathsf{sk3}}(impl)}_{\pi_{23}(impl)} \mathsf{sk2}.$$

By instantiating the model sk3 with the asymmetric channel implementation, we obtain a variant of the SKEME protocol (Figure 5) and the corresponding refinement of sk2.

	theories	defs	lemmas	lines	CPU time
Infrastructure	15 theories	65	660	5815	1 min 23 sec
Level 1	dh1, dh1n	44	117	1832	1 min 22 sec
Diffie-Hellman	dh2, dh3	56	137	2140	1 min 43 sec
SKEME/IKEv1	sk2, sk3	58	146	2548	4 min 41 sec
Total	25 theories	223	1283	12335	9 min 08 sec

Table 3. SPECIFICATION AND PROOF STATISTICS

4.6. Development statistics and discussion

Table 3 contains some statistics on our development, divided into four groups of theories. The first group includes our infrastructure theories for refinement and protocol modeling, including our Level 0 models. The other groups consist of the two theories indicated and the Level 3 instantiations. For each of these, we list the number of definitions and lemmas, the number of lines of the theory files, and the CPU time required to proof-check these theories. The measurements were made on a 2.6 GHz Intel Core i7 laptop with 8 GB RAM running Isabelle/HOL 2016-1.

Our development method proceeds stepwise and systematically across all four levels, whereby global security properties are first mapped to security guards on the exchanged information before being cast into actual network messages. We therefore provide a sizable infrastructure (*cf.* Table 3), which can be applied to other case studies. The state records and adversary events at all levels are also reusable. The protocol events follow a clear structure and only their concrete definitions depend on the protocol being modeled. The refinement mappings are simple and the required invariants are largely canonical, with similar proofs for different protocols. The resulting protocols are correct by construction.

5. Related work

There have been other proposals for developing security protocols by refinement using formalisms such as the B method [24], its combination with CSP [25], I/O automata [26], and abstract state machines (ASMs) [27]. None of these continue their refinements to the level of a full Dolev-Yao adversary. Either they only consider an adversary that is passive [26], defined ad-hoc [25], [27], or similar to our Level 2 adversary [24]. We previously proposed a systematic development method based on the four-level refinement strategy that we use in this paper [13], [14]. This prior work modeled a standard Dolev-Yao adversary, but not the advanced capabilities we consider here. Including these capabilities together with the possibility to model composed secrets and Diffie-Hellman key agreement required a major redesign of our framework.

Our advanced adversary model is based on the work of Basin and Cremers [8], [9]. They use the Scyther tool, which efficiently finds attacks, but for verification they sometimes had to bound the number of runs in their case studies. Our approach using theorem proving always guarantees security properties for the unbounded case. They consider additional forms of compromise such as the compromise of the test

run's owner to prove resilience against key compromise impersonation (KCI) [1] and randomness and state reveal used in some computational models [4], [6], [7]. While we could add these to our model, not all of them would be meaningful in our current framework. In particular, resilience against KCI typically requires asymmetric cryptography. However, our Level 2 channels are designed to enable both symmetric and asymmetric implementations, which prevents achieving resilience against KCI at Level 2. Supporting this would require a separation of symmetric/bidirectional and asymmetric/unidirectional channels.

A number of works have considered channels with security properties and their cryptographic implementations [20], [21], [28]–[30]. Bugliesi and Modesti [28] propose an Alice&Bob language extended with various forms of channels and a translation to standard Alice&Bob notation without channel abstractions. In [20], [21], the authors present two protocol models, an idealized one using channels (called ICM) and a cryptographic one (called CCM), where the channels are implemented using asymmetric cryptography, and prove that they simulate each other. However, the ICM simulates the CCM only under the assumption that messages can be fully decrypted [20] or that protocol are typable [21]. Our refinements proofs between Levels 2 and 3 use the weaker assumption that frames only contain payload messages (*i.e.*, no implementation material) and they are decoupled from the concrete channel implementations by only relying on our assumptions from Section 4.4.

The works [20], [29], [30] present compositionality results for multi-layered protocols, where an abstract application protocol relying on a channel with security properties must remain secure when instantiated with an entire lower-level protocol providing a secure implementation of that channel. Their soundness conditions ensure that there is no interference between the application protocol and the protocol implementing the channel and enables the independent verification of these two protocols. In contrast, we work in a simpler setup where each channel type is implemented by a single message. Accordingly, our assumptions on such implementations suffice to guarantee the desired channel properties at Level 3 and we can statically verify that concrete channel implementations satisfy these assumptions, without the need to consider protocol behaviors.

Abadi et al. [31] define a high-level process language with constructs for secure channels and compile it into a low-level language with cryptographic messages. They show a full abstraction result for their translation.

Datta et al. [32] 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. [33], [34] similarly refine protocols by transforming messages and propose specialized formalisms for establishing secrecy and authentication properties. While the instantiation and discharge of assumption is similar to our treatment of parametric channel implementations, their refinements do not involve fundamental changes of the abstraction level since one instantiates abstract operations on messages (at Level 3).

6. Conclusions

We have presented a protocol refinement framework that enables the development of key agreement protocols in the presence of a strong adversary. This required a major redesign and extension of our previous framework [13], [14], in particular, to handle composed secrets, Diffie-Hellman key agreement, and the strong adversaries.

There are some limitations in our framework that we would like to address in future work. As discussed in Section 5, additional forms of compromise could also be covered. However, in some of the related adversary models, only protocols using more complex equational theories, *e.g.*, with a group structure in the DH exponents, achieve security. Such equational theories would likely result in substantially more complex security proofs. One could possibly alleviate this problem by formalizing unification algorithms for decidable theories or by connecting Isabelle with external unification tools for such theories. Moreover, our current implementations of channel messages cannot use fresh nonces. These are useful, for instance, to construct a channel implementation using probabilistic or hybrid encryption or to build replay protection directly into the channels.

Acknowledgements. We are grateful to the anonymous reviewers for their helpful comments.

References

[1] M. Just and S. Vaudenay, "Authenticated multi-party key agreement," in *Advances in Cryptology - ASIACRYPT '96*, ser. Lecture Notes in Computer Science, K. Kim and T. Matsumoto, Eds., vol. 1163. Springer, 1996, pp. 36–49.

[2] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Advances in Cryptology - CRYPTO '93*, ser. Lecture Notes in Computer Science, D. R. Stinson, Ed., vol. 773. Springer, 1993, pp. 232–249.

[3] ——, "Provably secure session key distribution: the three party case," in *ACM Symposium on Theory of Computing*, F. T. Leighton and A. Borodin, Eds. ACM, 1995, pp. 57–66.

[4] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *Advances in Cryptology - EUROCRYPT 2001*, ser. Lecture Notes in Computer Science, B. Pfitzmann, Ed., vol. 2045. Springer, 2001, pp. 453–474.

[5] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Advances in Cryptology - EUROCRYPT 2000*, ser. Lecture Notes in Computer Science, B. Preneel, Ed., vol. 1807. Springer, 2000, pp. 139–155.

[6] H. Krawczyk, "HMQV: A high-performance secure Diffie-Hellman protocol," in *Advances in Cryptology - CRYPTO 2005*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Springer, 2005, pp. 546–566.

[7] B. A. LaMacchia, K. E. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Provable Security, First International Conference, ProvSec 2007*, ser. Lecture Notes in Computer Science, W. Susilo, J. K. Liu, and Y. Mu, Eds., vol. 4784. Springer, 2007, pp. 1–16.

[8] D. A. Basin and C. Cremers, "Know your enemy: Compromising adversaries in protocol analysis," *ACM Trans. Inf. Syst. Secur.*, vol. 17, no. 2, pp. 7:1–7:31, 2014.

[9] D. A. Basin and C. J. F. Cremers, "Modeling and analyzing security in the presence of compromising adversaries," in *15th European Symposium on Research in Computer Security, ESORICS 2010*, ser. Lecture Notes in Computer Science, D. Gritzalis, B. Preneel, and M. Theoharidou, Eds., vol. 6345. Springer, 2010, pp. 340–356.

[10] J. W. de Bakker, W. P. de Roever, and G. Rozenberg, Eds., *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop, 1989*, ser. Lecture Notes in Computer Science, vol. 430. Springer, 1990.

[11] M. Abadi and L. Lamport, "The existence of refinement mappings," *Theor. Comput. Sci.*, vol. 82, no. 2, pp. 253–284, 1991.

[12] J. Abrial, *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[13] C. Sprenger and D. A. Basin, "Developing security protocols by refinement," in *ACM Conference on Computer and Communications Security, CCS 2010*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds. ACM, 2010, pp. 361–374.

[14] ——, "Refining key establishment," in *IEEE Computer Security Foundations Symposium, CSF 2012*, S. Chong, Ed. IEEE Computer Society, 2012, pp. 230–246.

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

[16] J. Lallemand and C. Sprenger, "Refining authenticated key agreement with strong adversaries," *Archive of Formal Proofs*, February 2017, http://isa-afp.org/, Formal proof development.

[17] H. Krawczyk, "SKEME: a versatile secure key exchange mechanism for internet," in *1996 Symposium on Network and Distributed System Security, (S)NDSS '96*, J. T. Ellis, B. C. Neuman, and D. M. Balenson, Eds. IEEE Computer Society, 1996, pp. 114–127.

[18] D. Harkins and D. Carrel. (1998, November) The Internet Key Exchange (IKE). IETF RFC 2409 (Proposed Standard). Obsoleted by RFC 4306, updated by RFC 4109. [Online]. Available: http://www.ietf.org/rfc/rfc2409.txt

[19] L. C. Paulson, "The inductive approach to verifying cryptographic protocols," *Journal of Computer Security*, vol. 6, no. 1-2, pp. 85–128, 1998.

[20] S. Mödersheim and L. Viganò, "Secure pseudonymous channels," in *European Symposium on Research in Computer Security - ESORICS 2009*, ser. Lecture Notes in Computer Science, M. Backes and P. Ning, Eds., vol. 5789. Springer, 2009, pp. 337–354.

[21] M. Bugliesi, S. Calzavara, S. Mödersheim, and P. Modesti, "Security protocol specification and verification with AnBx," *Journal of Information Security and Application*, 2016, in press.

[22] G. Lowe, "A hierarchy of authentication specification," in *10th Computer Security Foundations Workshop (CSFW '97)*. IEEE Computer Society, 1997, pp. 31–44.

[23] U. M. Maurer and P. E. Schmid, "A calculus for secure channel establishment in open networks," in *European Symposium on Research in Computer Security - ESORICS 94*, ser. Lecture Notes in Computer Science, D. Gollmann, Ed., vol. 875. Springer, 1994, pp. 175–192.

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

[25] M. J. Butler, "On the use of data refinement in the development of secure communications systems," *Formal Asp. Comput.*, vol. 14, no. 1, pp. 2–34, 2002.

[26] N. A. Lynch, "I/O automaton models and proofs for shared-key communication systems," in *Proceedings of the 12th IEEE Computer Security Foundations Workshop, CSFW 1999*. IEEE Computer Society, 1999, pp. 14–29.

[27] G. Bella and E. Riccobene, "Formal analysis of the Kerberos authentication system," *J. UCS*, vol. 3, no. 12, pp. 1337–1381, 1997.

[28] M. Bugliesi and P. Modesti, "AnBx - security protocols design and verification," in *Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security - Joint Workshop, ARSPA-WITS 2010. Revised Selected Papers*, ser. Lecture Notes in Computer Science, A. Armando and G. Lowe, Eds., vol. 6186. Springer, 2010, pp. 164–184.

[29] A. Kamil and G. Lowe, "Understanding abstractions of secure channels," in *Formal Aspects of Security and Trust - 7th International Workshop, FAST 2010. Revised Selected Papers*, ser. Lecture Notes in Computer Science, P. Degano, S. Etalle, and J. D. Guttman, Eds., vol. 6561. Springer, 2010, pp. 50–64.

[30] S. Mödersheim and L. Viganò, "Sufficient conditions for vertical composition of security protocols," in *ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14*, S. Moriai, T. Jaeger, and K. Sakurai, Eds. ACM, 2014, pp. 435–446.

[31] M. Abadi, C. Fournet, and G. Gonthier, "Secure implementation of channel abstractions," *Inf. Comput.*, vol. 174, no. 1, pp. 37–83, 2002.

[32] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic, "A derivation system and compositional logic for security protocols," *Journal of Computer Security*, vol. 13, no. 3, pp. 423–482, 2005.

[33] D. Pavlovic and C. A. Meadows, "Deriving secrecy in key establishment protocols," in *European Symposium on Research in Computer Security - ESORICS 2006*, ser. Lecture Notes in Computer Science, D. Gollmann, J. Meier, and A. Sabelfeld, Eds., vol. 4189. Springer, 2006, pp. 384–403.

[34] I. Cervesato, C. A. Meadows, and D. Pavlovic, "An encapsulated authentication logic for reasoning about key distribution protocols," in *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005)*. IEEE Computer Society, 2005, pp. 48–61.