The Coinductive Approach to Verifying Cryptographic Protocols *

Jesse Hughes and Martijn Warnier

Computing Science Institute, University of Nijmegen
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands
{jesseh, warnier}@cs.kun.nl

Abstract We look at a new way of specifying and verifying cryptographic protocols using the Coalgebraic Class Specification Language. Protocols are specified into CCSL (with temporal operators for "free") and translated by the CCSL compiler into theories for the theorem prover PVS. Within PVS, the desired security conditions can then be (dis)proved.

In addition, we are interested in using assumptions which are reflected in real-life networks. However, as a result, we present only a partial solution here. We have not proved full correctness of a protocol under such loose restrictions. This prompts discussion of what assumptions are acceptable in protocol verification, and when practical concerns may outweigh theoretical motivations.

1 Introduction

Cryptographic protocols (also called "security protocols") give an abstract representation of a method for communicating securely over an open (insecure) network in such a way that goals such as *secrecy*, *authentication*, *integrity* or a combination of these conditions are satisfied. Verification of such protocols is (since [2]) an important subject of research where formal methods are applied. Their small size (usually just a few lines) and the difficulties in designing 'safe' protocols make them very suitable for formal analysis.

Of course, one must be careful in the formalization. The small expression of a protocol typically hides many implicit assumptions about the behavior of the principals, freshness, etc. Formalization of a protocol in a logic exposes implicit assumptions about the network, the behavior of participants, and so on.

The difficulty of analyzing security protocols is apparent from the history of the so called *Needham Schroeder public key authentication* protocol from [8]. This protocol, first published in 1976 and proved to be secure in 1989 (in [2]), contained a flaw which was finally found by Gavin Lowe (in [5]), 17 years after its original publication.

* To appear in the WADT 2002 proceedings, LNCS series

We propose a new way of analyzing cryptographic protocols, using the theory of coalgebras and hence "coinductive" reasoning[1] at its core. This approach is closely related to Paulson's inductive approach[11]. The difference between the two is conceptual. Paulson's model is fundamentally algebraic: he models the situation by an initial algebra of finite traces, and so relies on induction as his fundamental principle. We, instead, use a coalgebraic model (of infinite traces), and reason about invariants.

As a first case study, we analyzed the *Bilateral Key Exchange protocol* (BKE) [3], which we describe in Sect. 2. We built an informal model of the protocol, first by constructing an informal model of message passing in a network setting. This model includes the presence of a powerful spy, in the Dolev-Yao tradition [4].

When modeling the protocol, we took a decidedly liberal approach. We avoided certain assumptions present in other protocol verifications, particularly those that are not realistic, i.e., do not hold in real networks in which the protocol will be implemented. In particular, it is common to require that, if **A** and **B** are using a specific protocol, then the only messages they send are messages required by the protocol itself. Let us call this the *protocol message axiom.* But, in a natural setting, it is likely that, while **A** communicates with **B** over the network, she[2] is also holding other conversations with other participants (or even with **B**). So, it does not seem a realistic assumption.

The protocol message axiom may still be reasonable, if one has an a priori argument that it is a harmless assumption. That is, perhaps there is a good argument that, if there is a successful attack on the BKE protocol, then there is a successful attack during which **A** and **B** send only protocol messages. If so, then one ought to assume the protocol message axiom, since it simplifies the analysis. However, we do not have such an a priori argument. Consequently, we chose *not* to assume the axiom, so that our resulting correctness proofs are that much more persuasive.

The result was not wholly successful. If one drops the protocol message axiom, then one must carefully choose more reasonable axioms that allow the participants to act flexibly without behaving stupidly. These axioms include that the participants do not send messages with their private keys or with received secrets and so on. As well, one needs to decide when a received message contains a secret, so that the participant does not reveal it.

However, this is the minor part of the difficulty. We found that the resulting correctness proofs become very difficult indeed, because the reasoning involved requires subtler partitioning of classes of messages. This partitioning may be introduced via appropriate fixed point constructions, and adds considerable com-

[1] We use the term *coinductive* loosely here. We do not mean the principle that bisimilar elements of a final coalgebra are equal. Rather, we mean that our distinguished predicates are coalgebraic invariants, which are analogues to inductive predicates for algebras.

[2] It is customary in the literature on cryptographic protocols to refer to **A** and **B** as Alice and Bob.

plexity to the analysis. As well, the resulting proofs become subtle and difficult, and certainly are not amenable to automation.

In the end, we proved that, if the principals never send messages containing protocol secrets (the nonces, keys, etc., relevant to the protocol), aside from those required by the protocol, then these secrets will not become public (and, in particular, the Spy will never acquire a secret). However, we did not prove that this condition is met under reasonable assumptions about the behavior of the participants. In particular, one must show that the Spy can't compel one of the participants to reveal a secret to him (presumably by forgery). We believe that, indeed, this conclusion is provable, but found that the difficulty of the proofs was such that it became dubious that the proofs were worth the effort[3]. For purely practical concerns, perhaps the protocol message axiom ought to be adopted in formal security verifications, contrary to our approach.

This paper contributes a discussion of two orthogonal issues, then. First, we discuss coalgebraic models of security protocols. We advocate the Coalgebraic Class Specification Language, CCSL, for specifying protocols, exhibit the features of CCSL that make it attractive, and indicate how correctness proofs proceed. As a separate issue, we discuss a particular model for Bilateral Key Exchange, in which we have avoided the protocol message axiom. We explain the philosophical motivation behind this decision, and the large practical consequences as well. We offer the conflict between the philosophical and practical concerns as an open problem to be discussed.

The remainder of this paper is organized as follows: Sect. 2 gives a concrete example of a security protocol. Section 3 gives a informal overview of how we model the protocol without going into implementation details. Section 4 briefly describes the specification language CCSL [12], and how we implement the informal model using the specification language. Due to space restrictions, the technical discussions in Sects. 3, 4 and elsewhere are unfortunately curt. We hope that they are sufficient to convey the motivation and basic implementation of our work. Section 5 discusses the protocol message axiom and the effect of omitting it from our specification. Section 6 describes the relation between our approach and others and Sect. 7 ends with some conclusions.

2 An example Protocol

As an example of a security protocol, we consider the *Bilateral Key Exchange with Public Key Protocol* (see §6.6.6. of [3]). It is a simple protocol for distributing a session key between two principals **A** and **B**, who are also authenticated to each other. The session key can be used for secure communication thereafter.

We take the protocol as given, and do not criticize or alter it in our presentation. Thus, any redundant features are left unchanged in our analysis, etc.

[3] We think these problems are independent of our coalgebraic basis. They would also arise in Paulson's inductive approach if one omits the consequences of the protocol message axiom.

The usual abstract notation from the literature for such a protocol can be seen in Fig. 1.

1. $\mathbf{B} \rightarrow \mathbf{A} : B, \{N_b, B\}_{pk_a}$
2. $\mathbf{A} \rightarrow \mathbf{B} : \{Hash(N_b), N_a, A, K_{ab}\}_{pk_b}$
3. $\mathbf{B} \rightarrow \mathbf{A} : \{Hash(N_a)\}_{K_{ab}}$

Figure 1. Bilateral Key Exchange with Public Key Protocol

Informally, the protocol describes the following exchange.

1. **B** sends to **A** a message containing **B**'s identity, B, and an encrypted submessage including a nonce (i.e., a random number, assumed to be not guessable), N_b, and again the text B. It is encrypted with **A**'s public key.
2. When **A** receives the message, she decrypts it with her private key pk_a^{-1}. She responds by replying with a *Hash* of **B**'s nonce N_b, a new nonce N_a generated by **A**, the plain text message A and the session key K_{ab}. All are encrypted under **B**'s public key pk_b.
3. **B** receives and decrypts the message and sends a response to **A** containing a *Hash* of **A**'s nonce, N_a, encrypted using the session key K_{ab}.

We will use this protocol as a running example throughout this paper. We chose it for three reasons, (i) because it is simple, (ii) it is a hybrid protocol, meaning it uses both symmetric and asymmetric encryption, (iii) it has two different security goals, *authentication* and *secrecy.* These three features make the protocol (at least in our eyes) a useful example for a new verification approach.

3 Modeling

In this section we describe our model informally, giving basic concepts without technical details. We use the word “model” loosely here. Its meaning will stay informal for this section. Also, in this section, the coalgebraic features of our model are hidden. These features and a more technical description of our implementation in the Coalgebraic Class Specification Language [12,14] will be discussed in Sect. 4.

3.1 Security Model and Related Assumptions

We make a number of general assumptions and abstractions to form our *Security Model.* These should reflect the situations in which protocols are used.

Although the protocols are called *cryptographic* protocol we will not consider attacks on the underlying cryptography. Analysis of cryptographic protocols deals with the application of cryptographic primitives, not cryptography itself.

This so-called *Perfect Cryptography* assumption treats cryptographic primitives as black box processes that work as intended.

We consider a network with any number of participants and one attacker. The attacker is the most powerful one known to us; the spy from the Dolev-Yao model [4]. He is an active attacker who can intercept, redirect and alter arbitrary messages between principals. However, the spy cannot decrypt or encrypt messages with keys he does not know, so he has control over the network, but cannot defeat the encryption algorithms. The spy can also send and receive messages as a normal participant in the network.

Some of the other assumptions we use are not literally true about real-life networks and computations, but have practical justifications. In particular we use concepts such as "perfect" one way, collision free, hashes and "real" randomness of nonces.

There are a number of restrictions we omit that are common in the literature. For instance, we assume neither a fixed number of participants (as in [7]) nor a limited number of parallel protocol runs (as in [6,7]). Moreover, we avoid the following assumption.

> **Protocol Message Axiom:** If a participant **P** distinct from the Spy sends a message m, then m is required by the protocol.

For instance, in [11], the author explicitly includes axioms of the form:

> If *evs* is a trace containing "B says to **S** the message X," then *evs* may be extended by "**S** says to B' the message X'."

Because there is no other allowance that **S** (distinct from the Spy) sends a message[4], Paulson is committed to the protocol message axiom. To the best of our knowledge, all other approaches assume a similar restriction.

We chose to avoid this axiom because it is unrealistic. In a network, the participants send many different messages, the majority of which bear no resemblance to protocol messages. We wished to be clear on what assumptions on the behavior of participants are sufficient to conclude that the protocol will reach a correct state. For this end, we chose to adopt only those assumptions which are reasonably reflected in actual networks, or have a priori arguments for their acceptance. We do not have an argument for the acceptance of the protocol message axiom (although, in the end, practical concerns form a very strong argument).

Indeed, there is a simple argument against the axiom. Suppose that the initial message $B, \{N_b, B\}_{pk_a}$ in the Bilateral Key Exchange protocol is sent from **B** to **A**. Suppose as well that, prior to **A**'s receipt of same, the Spy sends **A** a message, containing $\{N_b, B\}_{pk_a}$ and a note asking for assistance decrypting the message. If **A** complies with this request, and sends a response back to the Spy, then clearly the protocol has gone awry.

[4] Although, Paulson does allow accidental loss via Oops events, these accidents do not really clarify what behavior among principals ensures success.

This is not to say that such behavior is reasonable on the part of **A**, but one would like protocol analyses to provide a more or less explicit description of what behavior *is* reasonable, i.e., what behavior ensures successful completion of the protocol.

Unfortunately, this aim is difficult indeed, and our present analysis did not wholly succeed in clarifying these issues. One must decide where feasibility supersede the motivations above.

3.2 Message Passing

We begin by describing a general setting in which principals pass messages in the presence of a powerful *Spy* (the Dolev-Yao spy described above). For now, messages can simply be seen as concatenations of *identifiers*, *nonces*, *hashes* and *keys*, all of which can be (multiple-times) encrypted. Example messages can be seen in the three steps from Fig. 1.

The model consists of states and transitions between states, where each state is intended as a temporal snapshot of the network. A state is fully characterized by the following:

- its action, that is, what has just occurred.
- the mode of the Principals (see below).
- its next state, i.e., what state follows the current state.

A state's action describes the Event that has occurred in that state. There are three Event types: (i) nothing happens, i.e., the system is idle; (ii) some Principal **P** sends a message m to another Principal **Q**; (iii) some Principal **P** receives message m. Note that the *Spy* is a Principal which can play the role of either **P** or **Q**.

A Principal's mode is fully given by (i) the public/private key pair a principal owns, (ii) his *knowledge* and (iii) the *beliefs* of the principal. Note that public and private keys are unique to each principal and that both *knowledge* and *beliefs* are state dependent, i.e., change as the system progresses (via next).

When we refer to an agent's knowledge, we mean the collection of messages with which he is familiar. A belief, for our purpose, is a statement of the form "P knows m". An agent ought to believe "P knows m" just in case he has seen evidence that m is indeed in **P**'s knowledge, but what counts as evidence depends on a specific protocol. Note: the objects of knowledge, for our purposes, are *messages*, while the objects of belief are *statements*. Knowledge does *not* consist of some variation of "true beliefs".

Given a state x, the next state (written x.next) is intended as the moment after x. Thus, our interpretation of "next" has a temporal flavor, yielding informal notions of prior, eventually, etc. It is constrained by the x.action and the mode of the principals at present. The constraints include:

- If x.action = idle, then the mode of each principal is unchanged in x.next.
- If, in x, **P** sent a message m to **Q** then

   1. The Spy learns m (see below).
   2. All other agents do not change their mode.
- If **P** receives message m, then
   1. **P** learns m.
   2. All other agents do not change their mode.

Notice that a model is a deterministic, but underspecified, transition system. Given a particular state x, there is a unique next state, x.next. Hence, the system is deterministic. However, even if one knows everything about state x, he will not know the action of x.next. Thus, it is underspecified.

Here we omit certain other constraints on our model. For instance, we require that the sender of a message already knows the message, and that sent messages are *Eventually* received[5]. Also particular protocols add additional constraints, including axioms regarding beliefs. More on this in Sect. 4.

When a principal **P** learns a message m (that is when **P**= *Spy* and m is sent, or when **P** receives m), then his knowledge increases. Clearly, after learning m, then **P** knows m. But he also knows the sub-messages of m, subject to the constraint that he only gains knowledge of an encrypted sub-message if he knows the appropriate key. As well, if m contains a key, then **P** may be able to decrypt other messages which he already knew. This complicated effect of learning is modeled by appropriate fixed points, about which more later.

3.3 Protocol

The general setting described in the previous section provides a foundation for the specification of particular protocols. Each protocol description includes also the following:

- rules governing reactions to certain messages;
- axioms governing belief updating;
- correctness conditions for the protocol.

The first item is complicated by the fact that we rejected the protocol message axiom in our model. In the presence of that axiom, one would include rules like: If a principal **Q** receives a message of the form $P, \{P, N_1\}_{pk_q}$, then he should respond with a message to **P** of the form $\{Hash(N_1), N_2, Q, K\}_{pk_p}$, where N_2 and K are fresh. Also, this second message is the only message which **Q** can send which contains N_1 (as a subterm).

However, a problem arises in that, **Q** may learn the nonce N_1 without receiving the full message $P, \{P, N_1\}_{pk_q}$. For instance, the *Spy* may intercept **P**'s message to **Q** and send simply $\{P, N_1\}_{pk_q}$. If **Q** does not treat N_1 as a secret with **P**, then the protocol will go awry. Therefore, our rule must be more restrictive. We require that, if **Q** receives a message m from which he can extract $\{P, N_1\}_{pk_q}$, then he will send an appropriate response to **P** and no other message from **Q** will contain N_1 as a subterm.

[5] But, in the meantime, the *Spy* can do whatever he likes.

As mentioned previously, beliefs are state and protocol dependent. For the Bilateral Key Exchange Protocol beliefs are modeled as follows:

– If, in some state x, protocol message 2 is received $(\{Hash(N_2), N_3, Q, K\}_{pk_p})$ by **P** and is consistent, then **P** believes in state x.next that **Q** knows the session key K.
– If, in some state x, protocol message 3 is received $(\{Hash(N_4)\}_K)$ by **Q** and is consistent, then the recipient of the message believes in state x.next that his correspondent knows the session key K.

We include these axioms so we may prove that, at the end of the protocol, both participants believe correctly that each knows the session key and hence they may communicate.

3.4 Correctness Conditions

Correctness conditions depend on the goals of a protocol. Looking again at the *Bilateral Key Exchange* protocol we are interested in two security concepts. First, we want to be certain that only the principals running the protocol learn the session key and no one else does, i.e., that the protocol ensures *secrecy* of the session key. The other interesting property is *authentication*, in this case mutual authentication. This means that both principals become certain of the identity of their correspondent.

In the end we want to prove something like the following: Given that the principals behave in a responsible manner and given the general security model, when an agent **B** starts the *Bilateral Key Exchange* protocol with another agent **A**, both agent **A** and agent **B** will eventually know the session key and both principal **A** and **B** will believe that each other knows the session key. This session key should be a shared secret between **A** and **B**. Explicitly, our notion of correctness requires the successful (but not timely!) completion of a protocol run.

Again, we stress that, at present, we have not proved such a strong condition given our weak axioms. We have shown that, if the participants never send any message containing the nonces N_a, N_b or keys pk_a^{-1}, pk_b^-1 and K_{ab}, aside from those messages required by the protocol, then K_{ab} will never become known to the *Spy*. This is an important step toward correctness, but there is much real work to finish the proof. This work involves subtle least fixed point constructions for classes of messages, among other considerations (see Sect. 4.3 for examples).

In the next section we will describe how we implemented this model in CCSL.

4 Implementation

The Coalgebraic Class Specification Language [12,14] provides a logical framework for writing specifications involving both algebras and coalgebras. Users can freely nest algebraic and coalgebraic specifications. This allows one to use algebraic models for inductively generated data types, e.g., the *static* structure in our implementation, and coalgebras for behavioral types, e.g., the *dynamic* structure of the implementation. Indeed, we call our approach "coinductive" to reflect our reliance on coalgebraic reasoning.

Let us return to the message passing structure from Sect. 3.2. We want to explicitly show how this description involves a coalgebraic structure. An implementation for the message passing context consists of a set X together with three functions:

$$\begin{aligned} \mathsf{action} &: X \longrightarrow \mathsf{Events} \\ \mathsf{mode} &: X \times \mathsf{Principal} \longrightarrow \mathsf{PrincMode} \\ \mathsf{next} &: X \longrightarrow X \end{aligned}$$

This is equivalent to a set X and a single function with type

$$X \longrightarrow \mathsf{Events} \times (\mathsf{PrincMode})^{\mathsf{Principal}} \times X$$

Hence we can view a model as a coalgebra, specifically a coalgebra for the functor $X \mapsto \mathsf{Events} \times (\mathsf{PrincMode})^{\mathsf{Principal}} \times X$.

More generally, one can view object oriented classes as coalgebras for a suitable functor. This is the key motivation behind CCSL, where a class specification is modeled by a coalgebra. CCSL provides the formal language for describing a class specification, and a compiler for translating specifications into logical theories.

One writes a formal specification in the language CCSL and compiles it into a theory for PVS [13,9] (or Isabelle [10]). This translation includes generating basic definitions, axioms and lemmas from the informal theory of coalgebras. For instance, CCSL creates the definitions for invariant predicate, homomorphism, etc., the proof principle of coinduction (in the case that our model is canonical), basic temporal operators for *Eventually* and *Always* as well as axioms (**S4**) and lemmas for these operators, and so on.

Our motivation for using CCSL, then, was threefold. First, a coalgebraic model seemed natural for security protocols. The dynamic features of a message passing system are easily translated to a coalgebraic structure. Second, the CCSL language is a well-developed formal setting for specifying coalgebraic models, and includes temporal operators useful for expressing correctness of security protocols. As well, CCSL supports mixing algebraic and coalgebraic specifications, which is particularly useful here. Third, the compiler creates PVS theories (in our use), so that we can exploit the speed and power of PVS in order to prove our theorems. This means that we can use a formal language built just for coalgebraic specifications to model the protocol, and use an existing theorem prover to prove our theorems. Also, the theory generated by CCSL goes well beyond what we include in the specification, since it includes the temporal axioms and coalgebraic features “for free”.

The fact that, in the end, we offer only a partial proof of the correctness of the protocol does not reflect on the appropriateness of CCSL and a coalgebraic approach generally. We still firmly believe that a coalgebraic model is perfectly appropriate and natural for a dynamic system, and that in particular CCSL offers a good setting for specifications. However, we must question whether our aim of avoiding the protocol message axiom is a practical decision. We are confident that the correctness proofs can be completed even without this axiom, but the resulting arguments are very long and difficult and the extra payoff largely theoretical at present.

4.1 Specification

In implementing the model we use two key features of CCSL, (i) modularization and (ii) inheritance.

By modularization, we mean that distinct features of our specification are treated separately. For instance, the features of a principal's mode (knowledge, beliefs, public/private key pair) can be largely separated from the message passing specification, although the latter depends on the former. More explicitly, we have a class specification, PrincMode, for the principal's mode, and another specification, MsgContext, for the message passing system. The effects of learning are segregated in the former, and changes there influence the latter only indirectly (via the mode : $X \times \mathsf{Principal} \longrightarrow \mathsf{PrincMode}$ method). This simplifies the creation of the specification, and allows one to change his implementation of knowledge-updating without explicitly changing the MsgContext specification (although, existing proofs will often need to be revisited).

The language CCSL includes an inheritance relation, so that one specification may inherit from another. This feature allows one to treat message passing generally, and inherit from this treatment for each individual protocol specification. In our work, we restrict our use of inheritance so that it preserves the models, in the sense that, given a particular protocol Prot, we have

$$\mathrm{Model}(\mathsf{Prot}) \subseteq \mathrm{Model}(\mathsf{MsgContext}).$$

In other words, all of the restrictions (i.e., axioms) that we have placed on the general setting apply to the particular setting of a given protocol. Thus, theorems proved about the general setting are inherited by the particular protocols as well, minimizing the amount of repeat effort.

Figure 2 shows the dependencies of the different classes, where *double* arrows ($\Longrightarrow$) show the inheritance relations. The *broken* arrow ($- - \succ$) shows that MsgContext depends on PrincMode. This dependency is an application of modularization in our specification.

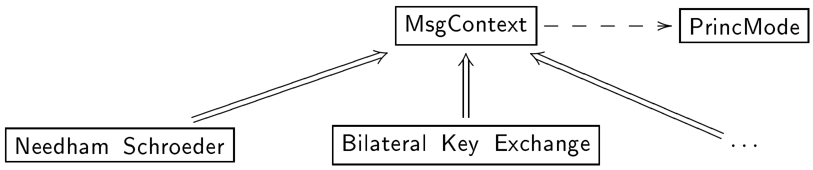

Figure 2. Theory dependencies

4.2 Temporal Operators in CCSL

As mentioned previously, the language CCSL includes the temporal *Always* ($\Box$) and *Eventually* ($\Diamond$) operators. In this section we will explain how these are defined.

Recall that the CCSL compiler defines an appropriate notion of *class invariant* for each class specification. The "always" operator is defined in terms of invariants. Specifically, $\Box P$ is the largest invariant which implies P, i.e.,

$$\Box P = \lambda(x : \mathsf{Self}) : \exists(Q : \mathsf{Self} \to \mathsf{bool})(Q(x) \wedge \mathrm{invariant}(Q) \wedge Q \Rightarrow P).$$

In our specification, the only method with non-constant codomain is next, which has type $\mathsf{Self} \to \mathsf{Self}$. This method has a natural temporal meaning, and this imposes a purely temporal interpretation on $\Box$. Indeed, we see that $\Box P(x)$ holds iff P holds at x and every state thereafter, so $\Box$ really is the *Always* (or "henceforth") temporal operator.

As usual, $\Diamond$ is defined as $\neg\Box\neg$. Therefore, $\Diamond P(x)$ holds iff there is some state "in the future" of x in which P holds, i.e., *Eventually* P.

The compiler provides, in addition to the definitions for $\Box$ and $\Diamond$, a number of lemmas about their behavior. In particular, the compiler proves that the operators satisfy $\mathsf{S4}$.

4.3 Messages and Learning

Messages are formally modeled by an *Abstract Data Type*. Explicitly, our model of messages is the least set S such that:

- For all $P \in \mathsf{Principal}$, $N \in \mathsf{Nonce}$, and $K \in \mathsf{Key}$, the syntactical objects $\mathsf{princ}(P)$, $\mathsf{nonce}(N)$ and $\mathsf{key}(K)$ are in S;
- If $m \in S$ then $\mathsf{hash}(m)$ is also in S;
- If $m_1 \in S$ and $m_2 \in S$ then $\mathsf{msgpair}(m_1, m_2)$ is also in S;
- If $m \in S$ and $K \in \mathsf{Key}$ then $\mathsf{symenc}(K, m)$ and $\mathsf{asymenc}(K, m)$ are also in S (representing symmetric and asymmetric encryption, resp.).

In other terms, our model of messages is an initial algebra, and in particular comes with a principle of induction.

We define two closure operators[6] on $\mathcal{P}(\mathsf{Message})$, the power set of Message. The first, $\downarrow$, (read: *downward closure*) is intended to capture extractability. That is, given $S \subseteq \mathsf{Message}$, a message m is in $\downarrow(S)$ just in case one can extract m from the messages in S (in particular, using only keys found in S). Explicitly, $\downarrow(S)$ is the least set satisfying

- $S \subseteq \downarrow(S)$;
- if $\mathsf{msgpair}(m_1, m_2) \in \downarrow(S)$, then so are m_1 and m_2;
- if $\mathsf{symenc}(K, m)$ and $\mathsf{key}(K)$ are in $\downarrow(S)$, then so is m;
- if $\mathsf{asymenc}(K, m)$ and $\mathsf{key}(K^{-1})$ are in $\downarrow(S)$, then so is m;

The upward closure, $\uparrow$, of a set S of Message is also given by a least fixed point condition. It consists of all of the messages that can be constructed from messages in S. We omit an explicit definition here, since it is analogous to $\downarrow$.

These two operators are used to axiomatize the effect of learning a message. Suppose that, in state x, an agent P knows exactly the set of messages S, and that he receives (or, if $P = \mathit{Spy}$, snoops) message m. Then, in state $x.\mathsf{next}$, the agent should know

$$\uparrow\downarrow(S \cup \{m\}).$$

(Note: given a set S, $\downarrow\uparrow\downarrow(S) \subseteq \uparrow\downarrow(S)$, and so $\uparrow\downarrow$ is idempotent.)

The fixed point constructions here are *not* sufficient to prove correctness of a protocol in the absence of the protocol message axiom. For that, subtler constructions are required, which we omit here for concern of space.

4.4 MsgContext

We can not show the complete specification of our model in this paper. Thus, in this section we will give a CCSL specifications of the constructs explained in Sect. 3.2.

Recall that we restrict the next operator according to the event and the mode of the principals at the current state. In the general MsgContext class we have defined an axiom action that does just this.

——Ccsl——

$\forall\ (P : \mathsf{Principal}), \forall(Q : \mathsf{Principal}), \forall(x : \mathsf{Self}), \forall(m : \mathsf{Message})$:
action:
  CASES x.action OF
    idle :
      $\forall\ (R : \mathsf{Principal})$:
        $x.\mathsf{next}.\mathsf{princmode}(R) = x.\mathsf{princmode}(R)$
    $\mathsf{sent}(\mathsf{P}, \mathsf{Q}, \mathsf{m})$:
      $\forall\ (R : \mathsf{Principal})$:
        $\neg(R = \mathit{Spy}) \rightarrow$

[6] The closure $\downarrow$ corresponds to Paulson's **analz** operator in [11], while $\uparrow$ corresponds to **synth**.

x.next.princmode(R) = x.princmode(R) $\wedge$
x.next.princmode(Spy) = x.princmode(Spy).learns(m) $\wedge$
x.princmode.knows(m) $\wedge$
($\Diamond$(λ(z : Self) :
receivedBy?(z.action) $\wedge$
receivedBy_rcp(z.action) = Q $\wedge$
receivedBy_msg(z.action) = m)
FOR{next}(x.next))
receivedBy(P, m) :
$\forall$ (R : Principal) :
$\neg$($R = P$) $\rightarrow$
x.next.princmode(R) = x.princmode(R) $\wedge$
x.next.princmode(P) = x.princmode(P).learns(m) $\wedge$
($\forall$(z : Self) : $\Diamond$(λ($x1$: Self) : $x1 = x$) FOR {next}(z) $\rightarrow$
($\Diamond$(λ($z1$: Self) :
sent?($z1$.action) $\wedge$
sent_rcp($z1$.action) = P $\wedge$
sent_msg($z1$.action) = m $\wedge$
$\Diamond$(λ($x1$: Self) : $x1 = x$) FOR {next}($z1$) $\wedge$
$\Diamond$(λ($z2$: Self) : $z2 = x$) FOR {next}($z1$) FOR
{next}(z) $\vee$
$\exists$($z1$: Self) : ($\Diamond$(λ($x1$: Self) : $x1 = z$) FOR {next}($z1$)$\wedge$
sent?($z1$.action) $\wedge$
sent_rcp($z1$.action) = P $\wedge$
sent_msg($z1$.action) = m $\wedge$
$\Diamond$(λ($x1$: Self) : $x1 = x$) FOR {next}($z1$))))

Notice that the general structure explained in Sect. 3.2 can be seen: the three types of Events are clearly distinct. There are also some additional temporal constraints. I.e., if a message is received in state x.next, then for all states z prior to x.next, there is a state $z1$ either prior to z or after z (but prior to x.next) such that message m was sent in state $z1$.

As well, we assume that when a message m is sent, it *Eventually* ($\Diamond$) will be received. At first glance, this seems to place a restriction on the ability of the *Spy*, since he cannot prevent a message from arriving. However, there is no time constraint on when the message will arrive, and so the *Spy* has as much time as necessary to interrupt the protocol. Thus, we do not feel that this assumption weakens the *Spy*.

4.5 PVS Theories and Proofs

There is a compiler for the CCSL language which can be used to translate the CCSL class specifications into theories for the PVS theorem prover [13,9]. In PVS then, we can prove a number of lemmas and of course the correctness conditions for different protocols we are interested in.

A theorem proving environment like PVS has advantages over informal mathematical arguments. In particular, PVS acts as a proof checker as the proof is being built, so that one is more confident of the final result. As well, with user-defined strategies tailor-made for the problems at hand, the theorem prover can take over much of the tedious work. Of course, such an environment comes with the usual difficulties of any formal system. Namely, proofs can become very long and tedious and the resulting proof may not be illuminating to a reader. In the case of protocol verification, however, one is interested in a proof only for establishing the theorem at hand, and not for its explanatory power.

5 A word about generality

There are two main features of our work which distinguish it from other approaches. First, as we have made clear, we take a coalgebraic approach to modeling message passing in general, and security protocols in particular. Second, we have followed the principle that an assumption about our model should be justified by an argument about its reasonableness or necessity. This principle is evident in the liberal assumptions regarding what messages may be sent.

These features are orthogonal, in the sense that (i) a coalgebraic model may certainly impose stronger assumptions than we do and (ii) other methods of verification may, in principle, weaken their assumptions as well. We anticipate that other methods will encounter similar complications if they choose to drop the protocol message axiom, however. It appears that this assumption comes with a real practical payoff, although it is unclear that the resulting proofs show the correctness of a protocol in a real-life setting. Again, one must decide at what point the practical concerns ought to trump the theoretical motivations.

In the end, we have proved an important part of the correctness of the Bilateral Key Exchange protocol. Namely, we have shown that, if **A** and **B** do not send messages containing N_a, N_b or K_{ab}, aside from the required protocol messages, then these secrets will never be known to the *Spy*. Note that, with the protocol message axiom, one needs merely to show that once the initial message is sent, then eventually the second and third messages will be accepted by their recipients to conclude that the protocol is correct. Even without the protocol message axiom, the direction for the remainder of the proof is clear, but it is not easy. When one allows a large body of legal messages, then it becomes difficult to express the relationship between the secrets and the body of messages as a whole. As well, when one allows liberal behavior on the part of the two correspondents, then restrictions that ensure the correct completion of the protocol become subtle. These proofs can be completed, but not without some sweat.

6 Related Work

There are a lot of different approaches described in the literature which deal with the verification of security protocols (see e.g. [1,2,6,7,11,15].) Generally speaking one can say that there are two kind of approaches. The ones that work (semi)automatic, these usually concentrate on some aspects of security while ignoring others, and in particular have some limitations such as the number of parallel sessions or the number of principals or protocol steps involved.

Then there are the more "total" approaches (most notably [11], see Sect. 6.1) which require a lot of user interaction but mostly do not have restrictions as the ones mentioned above. Clearly, our work must be placed in this latter setting. We want to stress that both kind of approaches are useful and are in many ways each other's complement.

6.1 Paulson's Inductive Approach

In his influential paper "*The Inductive Approach to verifying Cryptographic Protocols*" [11], Larry Paulson describes an approach for analyzing security protocols using the theorem prover Isabelle [10]. Although our approach is similar to his, there are some notable conceptual differences (aside from the practical differences in the specifications, including his assumption of the protocol message axiom).

Paulson's models are inherently algebraic, rather than coalgebraic. He considers the set of finite traces for a protocol. This set can be given by a least fixed point construction, i.e., by an initial algebra.

His basic proof principle is induction over traces of events (evs). To prove P always holds, he shows that

$$P[] \tag{1}$$

holds, and if

$$P(evs) \Rightarrow P(ev\#\ evs). \tag{2}$$

i.e., P holds for the empty trace of events [] and if P holds for some trace evs, P has to hold for all traces $ev\#\ evs$ containing one event ev more.

This is analogous to showing that P is an invariant, in the coalgebraic sense. The main theoretical difference is that we consider finite and infinite traces as models, while Paulson considers finite traces. Moreover, since Paulson relies on the inductive proof principle, he is restricted to canonical algebras as models. Because we do not assume finality of our coalgebraic model, we have a more general approach. In particular, Paulson's inductive models can be viewed as models in our coalgebraic setting (since initial algebras are also coalgebras), at least given an appropriate specification.

7 Conclusion

Our protocol verification departs from other approaches in two distinct features: its explicitly coalgebraic (and temporal) foundation, and the weakness of our axiomatization of the protocol. The former feature is an illustration of an application of abstract mathematics to verifications, making use of coinductively specified structures. The latter feature was motivated by a desire to strengthen the persuasiveness of the correctness proofs, by avoiding restrictions that did not hold in real networks. In particular, we avoided the so-called protocol message axiom.

We have argued that coalgebras are particularly useful for modeling dynamic systems. We think that the separate specification language (allowing also modularization and inheritance) as well as the higher abstraction in the proofs are other attractive features of our approach.

It would be useful to compare coinductive versus inductive reasoning on even terms, which we have not done here. One should adapt Paulson's axiomatization to a coalgebraic setting (or vice versa). With such a study, one could see the advantages and disadvantages of coalgebraic/algebraic conceptual frameworks in cryptographic protocol analysis.

We ran into considerable difficulties in proving the correctness of our sample protocol without the protocol message axiom. We have shown that, if the participants **A** and **B** transmit messages as called for by the protocol, then the Spy never learns the session key (or any of the nonces or private keys). We have not proved that the Spy can't compel **A** or **B** to send a message to him, revealing the session key or nonces. We also have not shown that beliefs which are acquired during the protocol run are true.

These partial successes emphasize the importance of the protocol message axiom, which has not been previously considered. While we have, we believe, strong theoretical motivations for omitting the protocol message axiom, we found perhaps stronger practical reasons to accept it. This negative result prompts questions about protocol correctness proofs. At what point ought practical concerns allow one to accept assumptions which are demonstrably false in natural settings, and which are not apparently "harmless"? We do not find an obvious answer to this fundamentally philosophical question.

Acknowledgments

We want to thank Bart Jacobs for suggesting the research topic and helping us on our way using CCSL, and Hendrik Tews for explaining CCSL constructs and repeatedly modifying the CCSL compiler to satisfy our whims.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, April 1997.
2. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proc. Royal Soc.*, Series A, Volume 426:233–271, 1989.
3. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature, version 1.0, 1997. available at URL `http://www-users.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.

4. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(6), 1983.
5. Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.
6. Gavin Lowe. Casper: A compiler for the analysis of security protocols. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.
7. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *8th ACM Conference on Computer and Communication Security*, pages 166–175. ACM SIGSAC, November 2001.
8. R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
9. S. Owre, J.M. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. on Softw. Eng.*, 21(2):107–125, 1995.
10. L.C. Paulson. *Isabelle: A Generic Theorem Prover.* Number 828 in Lect. Notes Comp. Sci. Springer, Berlin, 1994.
11. L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journ. of Computer Security*, 6:85–128, 1998.
12. J. Rothe, H. Tews, and B. Jacobs. The coalgebraic class specification language CCSL. *Journal of Universal Comp. Sci.*, 7(2), 2001.
13. N. Shanker, S. Owre, J.M. Rushby, and D. Stringer-Calvert. *PVS prover guide*, 1999. Version 2.3.
14. Hendrik Tews. *Coalgebraic Methods for Object Oriented Specification*. PhD thesis, Technical University of Dresden, October 2002.
15. F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1), 1999.