

Proving Authentication Properties in the Protocol Derivation Assistant

Matthias Anlauff,¹ Dusko Pavlovic,¹
Richard Waldinger,² and Stephen Westfold¹

¹ Kestrel Institute, Palo Alto, California, USA,
{ma,dusko,westfold}@kestrel.edu

² SRI International, Menlo Park, California, USA,
waldinger@ai.sri.com

Abstract. We present a formal framework for incremental reasoning about authentication protocols, supported by the Protocol Derivation Assistant (PDA). A salient feature of our derivational approach is that proofs of properties of complex protocols are factored into simpler proofs of properties of their components, combined with proofs that the relevant refinement and composition operations preserve the proven properties or transform them in the desired way.

In the present paper, we introduce an axiomatic theory of authentication suitable for the automatic proof of authentication properties. We describe a proof of the authentication property of a simple protocol, as derived in PDA, for which the the proof obligations have been automatically generated and discharged. Producing the proof forced us to spell out previously unrecognized assumptions, on which the correctness of the protocol depends.

PDA has support for collaboration and tool integration. It can be freely downloaded from [5].

Keywords: Protocol Derivation, Authentication, Incremental Proof, Security Reasoning.

1 Introduction

Background: Reasoning about security. The research area of security has generated a surprisingly wide range of models and approaches. Even the basic paradigm of security comes in three different flavors: computational (initiated by Diffie and Hellman [16]), information-theoretic (based on Shannon’s work [39]), and symbolic (due to Dolev and Yao [17]). Interestingly, the most abstract (and hence the least precise) model is the most recent, and efforts to relate the three paradigms are of an even later date [2, 28].

Even within the area of symbolic modeling, one encounters a remarkable diversity of logics [8, 13, 15], models [1, 10, 21, 38], and tools [7, 27, 29, 32, 33]³, combining generic and domain-specific methods in various ways. As we are about

³ The references are selected only to illustrate the diversity, with no attempt to reflect the field, or even the authors’ own taste.

to propose yet another framework for reasoning about security protocols, this invites the questions *Why such diversity? Why is security so hard to pin down? And what will be added by our work?*

The answers arise from a new, as yet unexplicated, paradigm of computation that is the driving force behind a conceptual upsurge of security research.

Problem and objective: Protocols as computation. Since its inception, computer science has been preoccupied with such state machines as Turing machines and automata. Consequently, computations were defined as possible executions, i.e., sequences of actions, and reasoning about computation has been in terms of predicates over such sequences. A succinct expression of this approach is the slogan “Programs are predicates” [23], which means that “Each program is interpreted as the strongest [predicate] describing its observable behavior on all its possible executions” [24]. Program correctness is then established by proving that, for all possible executions, “bad things” will not happen and “good things” will happen, which is guaranteed, respectively, by the *safety* and *liveness* properties of the program in question [26, 3].

With the advent of the Internet, and of computer networks in general, this simple paradigm of computation becomes insufficient. The interesting computational processes nowadays do not occur within a state machine, but are distributed through interactions of many state systems, which may or may not be observable. Such interactions are specified by protocols. Protocol computation thus consists of the information flows that connect local executions, i.e., of information flows and information boundaries. In order to program and control such computation, we need to be able to construct sound derivations of global properties from local observations. This is where fundamentally new problems in security engineering begin.

The semantic distinction between reasoning about protocol security and more traditional program correctness is that security properties are not predicates over the possible executions, but over the possible information flows. Like correctness properties, they can be factored into the statements that bad things will not happen and good things will happen. Indeed, secrecy (privacy, anonymity. . .) says, roughly, that undesired information flows do not happen, whereas authenticity (integrity, non-repudiation. . .) says, roughly, that the desired information flows do happen. The novelty here is that the positive and the negative properties dynamically depend on each other: every secret needs to be authenticated, and all authentications are based on (broadly construed) secrets. Information flows are assured by information boundaries, and vice versa.⁴

The problem arising from protocol computation is that traditional logical systems are not made to support distributed reasoning, which is its very essence. Our objective is to develop modeling and logical methodologies, tools, and interfaces to facilitate interaction with the novel and unintuitive logical situations of distributed reasoning. Such tools and interfaces can perhaps be construed as extensions of our natural abilities for centralized reasoning, just as the dashboards

⁴ *Encapsulating* secrecy and authenticity reasoning into separate logical modules [11, 34] brings this logical interplay to the surface.

and controls of cars and airplanes can be viewed as extensions of our senses and motor functions, allowing us to move at speeds and altitudes for which we are not naturally equipped.

Background and approach: Protocol derivations. To describe a protocol, one usually specifies (i) its process model, (ii) its logical properties, and (iii) its incremental development: conceptual components, versions, predecessors, and descendants. The Protocol Derivation System (PDS) and the Protocol Derivation Assistant (PDA) extend over these three dimensions as well.

The idea to analyze protocols in the space spanned by their process models and their logical properties goes back to [19, 18], where Protocol Composition Logic (PCL) was used to attach Floyd-Hoare-style annotations to process expressions describing protocol roles. The idea that this should be done incrementally, beginning with simple protocol components and simple security properties, which are then refined and composed towards more complex structures, has been developed in [14, 12, 13]. The general approach to analyzing complex computational interactions, by refining and composing process descriptions together with their logical annotations, was based on earlier work on *evolving specifications*, where annotations were attached to code of adaptable software modules [35, 4, 36]. The Protocol Derivation Assistant inherits not only the basic design principles, but also a part of the code base from an earlier tool prototype, built to support evolving specifications.

However, the incremental approach to security imposes essential new challenges, not tackled in software development at large, as explained e.g. in [13, 30]. Even if factored into small incremental steps, the complexity of reasoning, and even of the notations, can very quickly grow out of proportion. One source of complexity can be subtle semantical interactions of the process model and the logical formalism within the derivational formalism. Soundness becomes a difficult problem. Mechanizing complex derivational rules even harder.

One way to mitigate this problem has been to choose a process model closely tied to a convenient logical signature. The Protocol Derivation System [30, 11] is a result of an effort in that direction. Since authentication proofs largely consist of reasoning about the order of actions, using a process model based on partially ordered multisets (pomsets) [37], rather than a reaction-based process calculus, turns out to dramatically simplify many authenticity proofs. Moreover, choosing a logical signature (as presented below) that directly reflects the features of this model reduces the soundness of the rules and axioms to a matter of inspection. A further simplification is achieved by separating the authenticity proofs, realized through positive conclusions about the order of actions, from the secrecy proofs, deferred and *encapsulated* into open assumptions [11], to be discharged in a separate logical module, through negative conclusions about computability of terms [34].

With these simplifications, we took up the the task of providing automated support for protocol derivations through the Protocol Derivation Assistant. Note that the essence of this tool is *not* to automate theorem proving about security properties, nor to support graphic specifications of protocols. Its essence is to

support *incremental* specifications of protocols, annotated with their security properties.

Reuseable Taxonomy of Protocols and Properties. PDA allows abstract protocols to be defined and then refined and combined to derive a taxonomy of protocols. Each refinement step or PDA rule application can use the authenticity theorems of its input protocols to help prove authenticity for the resulting protocol. This incremental approach provides natural structure to proving authenticity for protocols. The derivation trees of protocols in PDA effectively provides a structured library of protocols and authenticity theorems for the protocol designer to reuse when designing new protocols and proving authenticity.

PDA provides tool support for the derivational approach to protocols presented in this paper. We will give here just a brief overview of the capabilities of PDA; a full description of its functionality is beyond the scope of this paper. For further information please see [5]. The design of PDA reflects the basic ideas of the derivational approach to protocol design by providing (i) a rich, graphical user interface for entering protocol derivations, (ii) support for refining models that correspond to these protocols, and (iii) automated support for incrementally proving security properties of the protocols and their models.

PDA provides a rich set of features that support the developers in their tasks of creating and manipulating protocols and protocol derivations. From abstract protocol definitions PDA users can derive more concrete protocols using instance creation, where function parameters are instantiated with more concrete functions. In parallel with the (graphical) protocol derivations, PDA supports the specifications and refinement of protocol models. In order to make use of the model refinement support in PDA the user specifies the semantics of functions used in the protocol alongside with basic axioms in the spec-part of the protocol, which is accessible from the graphical user interface. Using this and the process graph defined for the protocol, PDA generates proof obligations for authenticity properties. These proof obligations can be discharged (or not) from within the PDA-GUI using prove commands. When the proof goes through, the corresponding conjecture is transformed into a theorem for refinements of the protocol for which it has been proven. This concept manifests the power of the incremental approach, because in subsequent derivation steps these proofs do not need to be repeated, and thus decrease the complexity of the correctness proofs to a level where they are manageable and in most cases comprehensible.

Outline of the paper. The presented derivation is a detailed elaboration of a part of the analysis of the GDOI protocol presented in [30]. Section 2 goes into more detail about support functionality and architecture of PDA. We give an overview of the derivation in section 3, showing how the proof of authentication of the final protocol is built up from proofs at each derivation step. In section 4 we present the authentication theory we have developed in Specware for expressing the protocol obligations, so that they can be proved using the SNARK theorem prover. In section 5 we describe the authenticity obligations that arise in the derivation in terms of this theory, presenting in detail the proof of how the abstract challenge response protocol can be realized using a hash function.

This includes a description of how authenticity obligations are generated automatically for a protocol, and a discussion of theorem proving issues that arise in this work. Finally, we present conclusions and discuss future work.

2 PDA– The Protocol Derivation Assistant

The Protocol Derivation Assistant (PDA) provides tool support for the derivational approach to protocols presented in this paper. This section gives a brief overview of the capabilities of PDA; a full description of its functionality is beyond the scope of this paper. For further information please see [5].

2.1 Support Functionality

As already mentioned in the previous section, the design of PDA reflects the basic ideas of the derivational approach to protocol design by providing (i) a rich, graphical user interface for entering protocol derivations, (ii) support for refining models that correspond to these protocols, and (iii) automated support for incrementally proving security properties of the protocols and their models. We will briefly sketch these three aspects in the following.

Protocol Derivations. Being implemented as an application on top of the Eclipse platform and its Graphical Editing Framework (GEF) [20, 22], PDA provides a rich set of features that support the developers in their tasks of creating and manipulating protocols and protocol derivations. Protocols are specified using a graphical editing pane by drawing the desired run of the protocol similar to the representation of protocols found in academic research articles. Figure 1

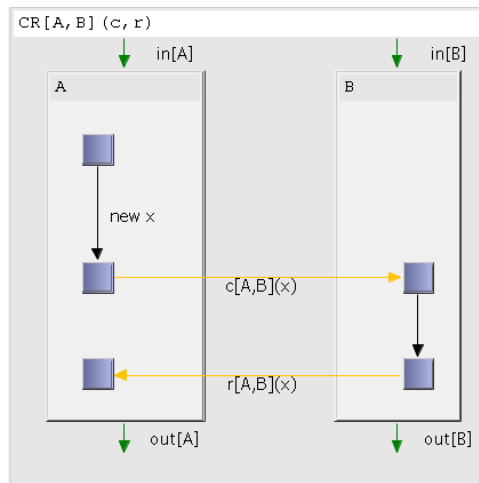


Fig. 1. Specifying a protocol in PDA using *desired-run* notation

shows the basic challenge-response protocol: protocol definition given as label

“ $\text{CR}[A,B](c,r)$ ” specifies the two roles initiator (A) and responder (B) as well as the generic challenge and response functions (c and r , respectively). From protocol definitions like this PDA users can derive more concrete protocols using instance creation, where function parameters (e.g., c and r in Figure 1) are instantiated with more concrete functions. PDA then automatically generates the protocol graphics with the values for the functions substituted in messages and internal actions. For more sophisticated transformations, PDA offers the concept of *rules*, which can be used to specify arbitrary transformations and/or compositions of protocols in order to construct a refined protocol out of already derived or defined ones. This concept of rules is very powerful and has already allowed for expressing crucial transformation steps in the derivations of popular security protocols like GDOI [30], MQV[31], etc. PDA also provides means to keep the protocol derivations organized by allowing the user to split larger derivations into multiple files and by defining a specialized derivation browser that displays the structure of the derivations regardless of their division into multiple diagrams. In addition, the user can define filters called *working sets* on top of files and folders; this allows for different derivations living in the same workspace and possibly sharing partial derivations.

Model Refinement and Automated Support. In parallel with the (graphical) protocol derivations, PDA supports the specifications and refinement of protocol models. In order to not dictate a specific logic and interpretation of the protocols, the core PDA system defines interfaces for plugging in arbitrary specification frameworks. In its current, version PDA is shipped with a plugin for Kestrel’s Specware specification language [25], which provides powerful functionality to support model refinement. The Specware-plugin uses the builtin S-expression generator that transforms the protocol graphics into S-expression text.⁵ In order to make use of the model refinement support in PDA the user specifies the semantics of functions used in the protocol alongside with basic axioms in the spec-part of the protocol, which is accessible from the graphical user interface. Using this and the process graph defined for the protocol, the Specware-plugin generates proof obligations for authenticity properties. These proof obligations can be discharged (or not) from within the PDA-GUI using prove commands, which trigger calls to the SNARK theorem prover [40] integrated into the Specware system. When the proof succeeds, the corresponding conjecture will be transformed into a theorem for refinements of the protocol. Figure 2 shows a screenshot involving the use of the theorem prover in a protocol derivation.

2.2 The Architecture of PDA

Figure 3 sketches the architecture of the PDA tool. The user enters protocol definitions and derivations in the graphical editor, which has a rich set of features to ensure the scalability of the approach. Most prominently, the graphical nodes representing protocols, agents, rules, etc. can be collapsed and expanded as

⁵ The format of the S-expression generated by PDA is the result of an effort to integrate with several other security related tools from Mitre, SRI, and others.

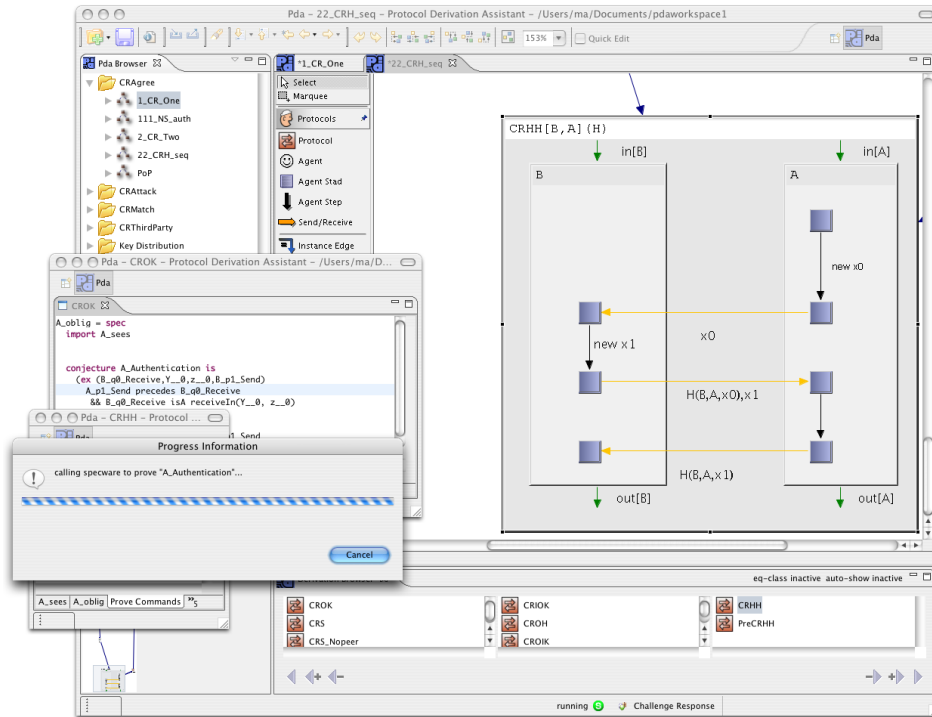


Fig. 2. Running PDA session involving theorem prover invocation

needed, which greatly improves the readability of complex derivation diagrams. While drawing the nodes and edges that make up protocols or derivations, the user gets some live feedback that prevents him/her from adding nodes and edges that are not permitted. For instance, if one side of a send/receive edge has been attached to a state in an agent node, then the user interface makes it impossible to attach the other end of the edge to a state within the same agent. The labels attached to protocols, internal actions, send/receive term, agents and other elements of the derivation are subject to corresponding syntax and semantics rules that are implemented in the parser and static analyzer. If the user makes an error on one of these labels, the graphical editor displays a visual feedback next to the place where the error has been detected. The derivation engine is responsible for performing instantiations and transformation operations, and for providing the result of these operations to the user as new nodes in the graphical editor pane. For example, for an instantiation, the user only enters the definition term for the refined protocol, the process graph of the instance is created automatically by the derivation engine component. All objects involved in the protocol derivations are stored in a database in order to allow for efficient access and

update operations. In its current version, the database is built into PDA, but future versions will provide the possibility to use server-based databases.

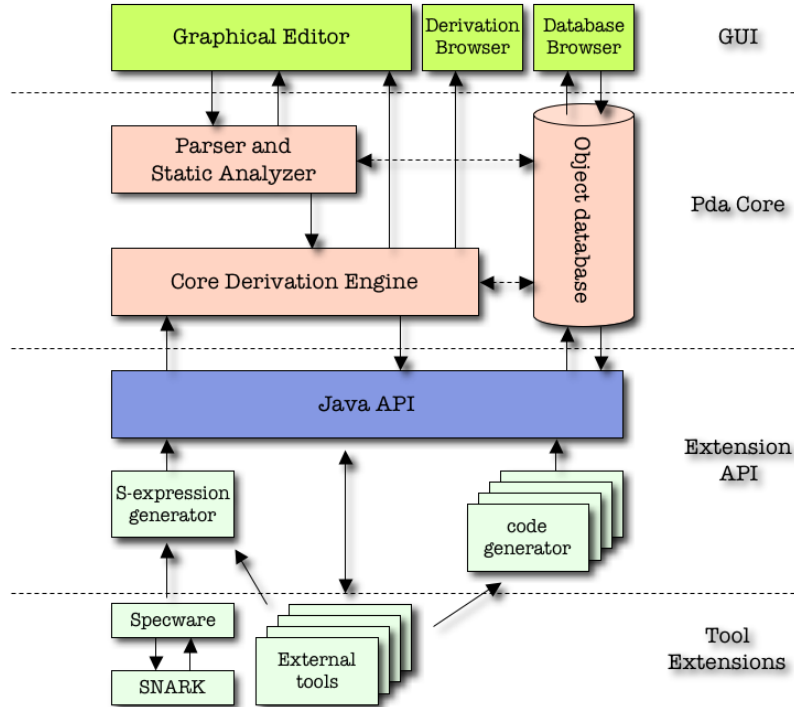


Fig. 3. PDA architecture

PDA is also designed to be an integration platform for security-protocol related tools. PDA provides an API that allows Java developers to write plugins for PDA. The API gives access to the internal data structures of the protocols specified by the user and/or loaded into the PDA-database. In order to make PDA also available for extension on a non-Java basis, PDA comes with an S-expression generator that translates the graphics of the protocols and the attached specifications into an S-expression format. The Specware-plugin mentioned earlier makes use of this interface and provides itself a user interface that allows the user to attach model descriptions to protocols. Other code generators can be defined as needed, for instance one for generating executable agent code from the protocol descriptions. Other tools can plug into PDA by either using one of the code generators or by directly using the Java API.

3 Derivation of a Mutual Authentication Protocol

The derivation of the example mutual authentication protocol in PDA is shown in Figure 4.⁶ We begin with an abstract one-way challenge-response protocol (CR), in which agentA (Alice) authenticates agentB (Bob). This is then instantiated to use a hash-based response function (CRH). A copy of this protocol is made with switched roles, so that Bob authenticates Alice (CRH_{op}). These two one-way authentication protocols are then composed sequentially using the rule $CompSeq$ to obtain a mutual authentication protocol SEQ . The resulting protocol is simplified by merging the last step of the first protocol with the first step of the second protocol using the rule Glu . Finally, a simple protocol transformation is made to bind the two one-way authentications, introducing Bob’s challenge into the hash so that Alice can be sure the challenge is from Bob ($CRHH$).

Each protocol has an associated specification ($spec$) that characterizes the functions in the protocol. In protocol CR there is an axiom that gives the property that the challenge function c and response function r must obey for the authentication obligation to be satisfied. When we instantiate a protocol we give a translation for one or more of the function symbols of the parent protocol. This translation defines an *interpretation* between the spec of the parent and the spec of the instance. Such interpretations between a source and target specs have the property that if the axioms of the source spec are theorems when translated into the target spec, then all theorems of the source spec are theorems when translated to the target [9]. This is a key property of the Specware system we use for writing these specs [25], and it allows us to factor the proof of authenticity. In particular, after an authentication conjecture has been proved in a protocol, its translation is true in an instance protocol provided the translated axioms are proven true. For example, having proven that authenticity in CR follows from the axioms of its spec, we only need to show that the translation of the axioms in CRH follow from the properties of the hash function, and then we know that the authenticity property holds in CRH .

PDA rules restructure protocols so that authentication properties are not, in general, preserved. However, typically, structure from the inputs appears in the outputs, so there is opportunity to use authenticity properties of the inputs to help prove authenticity in the output. To increase the applicability of authenticity theorems proved, we make them self-contained, with all assumptions explicit in the premise, so they can be safely applied in different contexts, and we generalize them to increase their applicability.

4 A Machine-Oriented Theory of Authentication

We have endeavored to develop a theory rich enough to express the concepts of the Pda system yet restricted enough to allow efficient automatic treatment of the proof obligations the system generates. The theory is expressed in the logical language of Specware, and proofs are discovered by the theorem prover SNARK[40], which is invoked through Specware’s theorem-prover interface. We

⁶ We use A , B , c , r and H for *agentA*, *agentB*, *challenge*, *response* and *hash* for brevity in the figure.

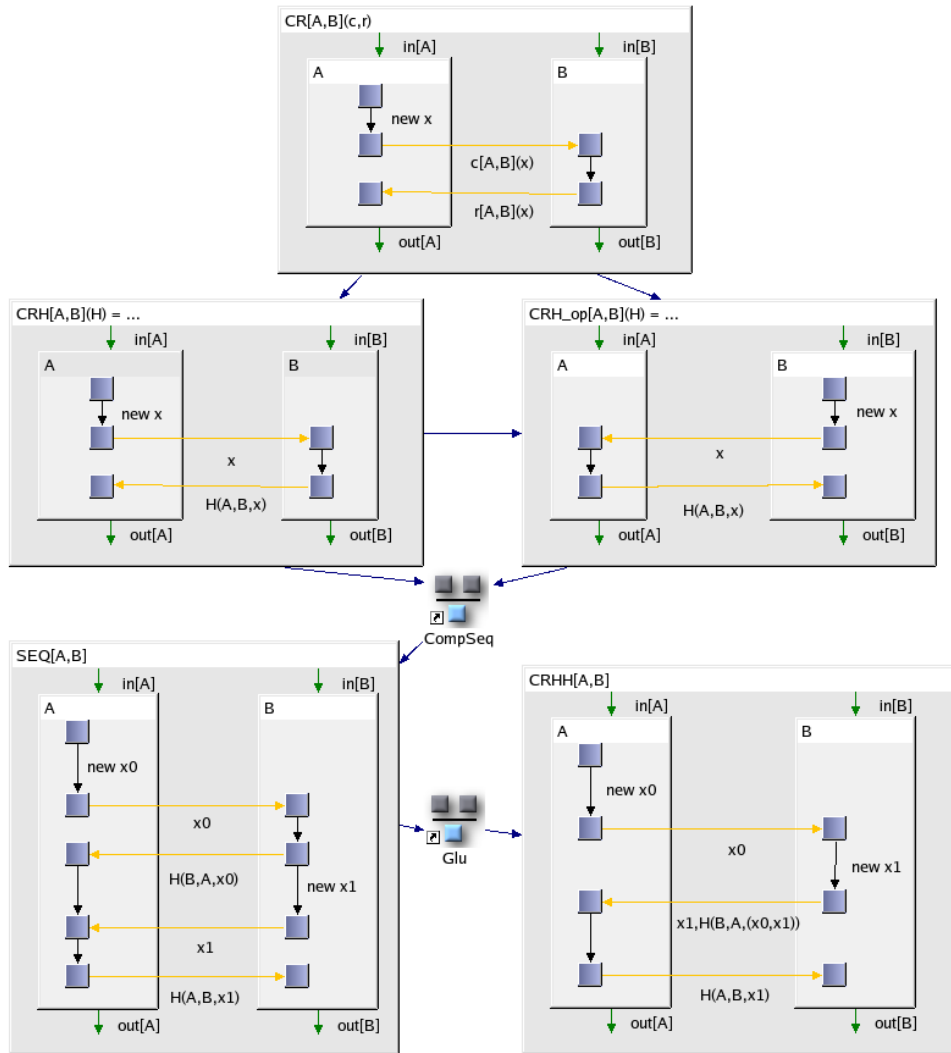


Fig. 4. Screenshot of derivation of example mutual authentication protocol in PDA

intend ultimately to use the same theory to automatically construct attacks on vulnerable protocols.

We found the mechanization effort instructive in that it forced us to spell out assumptions that escape notice in hand verification, however formal. The fact that the proofs are checked quickly and automatically gives us the freedom to explore alternative formulations without fear of losing the authentication properties.

4.1 The Theory

The theory defines the sending and receiving of messages, the temporal relationship between these actions, and the properties of nonces and cryptographic hashing. We present it here anecdotally, introducing the sorts, the fundamental relations and functions, and samples of axiomatization. The full theory, in the Specware notation, is on display at

<http://www.kestrel.edu/software/pda/APWW06/>. In some cases, the formulas in the theory have been altered to simplify the explanation.

4.2 Sorts

Our theory discriminates among three fundamental sorts of entities.

- **Agents:** The human or mechanical entities that send or receive messages. They include the principals of the protocol and also potential intruders.
- **Actions:** The sending and receiving of messages. We distinguish between different sendings or receivings of the same message by the same participant.
- **Terms:** The content of messages including text, nonces, and hashed text.

4.3 Basic Constructs

Here are the basic functions and relations on which the theory is based.

The pair function. The pair function $\langle term1, term2 \rangle$ combines two terms; for any terms $t1$ and $t2$, $\langle t1, t2 \rangle$ is another term. We assume that the pair function is one-to-one in both of its arguments; in other words, we have the axiom

$$\forall(t1, t2, tt1, tt2) \langle t1, t2 \rangle = \langle tt1, tt2 \rangle \Rightarrow t1 = tt1 \wedge t2 = tt2.$$

The inside relation. The relation $term1$ *inside* $term2$ holds if $term1$ is a part of $term2$. It is not assumed that terms are strings or that inside is precisely the substring relation; for instance, we will assert that a term is inside its hashing, even though the hashing may be smaller than the term. If a term is inside another, we shall also say that the latter term *includes* the former.

The inside relation is a reflexive ordering; a term is inside itself. The pair of terms includes both component terms; in other words, we have the axiom

$$\forall(t1, t2) t1 \text{ inside } \langle t1, t2 \rangle \wedge t2 \text{ inside } \langle t1, t2 \rangle.$$

The precedes relation. We say $action1$ precedes $action2$ if $action1$ occurs strictly earlier than $action2$. This relation is not assumed to be total; in other words, there can be two actions whose temporal order is indeterminate or unspecified. If one action precedes another, we shall also say that the latter relation follows the former. And we shall say that the former relation occurs before the latter, and that the latter occurs after the former.

The precedes relation is a strict well-founded ordering. In other words, an action does not precede itself and it is impossible to have an infinite sequence of actions that go backwards in time; there is no sequence $action0$, $action1$, $action2$, ... such that $action0$ follows $action1$, $action1$ follows $action2$, and so on, each action temporally following the next one in the sequence.

The send and receive relations.

$send(action, agent, term)$

holds if $action$ is a sending of a message $term$ by $agent$. Similarly,

$receive(action, agent, term)$

holds if $action$ is a receiving of the message $term$ by $agent$. It is assumed that $term$ is the entire message. The relations are one-to-one, in the sense that each sending and receiving has a unique agent (the sender or receiver) and a unique term, the text of the message.

At this stage in the development of the theory, we ignore the alleged sender and the intended recipient of a message. We imagine that when a message is sent, all agents can see it and none can be sure who sent it.

The send and receive actions are assumed to obey the following axiom Rcv , which asserts that if a message is received, the message was previously sent.

$$\begin{aligned} & \forall (receiving, receiver, term) \\ & \quad receive(receiving, receiver, term) \\ & \Rightarrow \exists (sending, sender) \\ & \quad send(sending, sender, term) \wedge sending \text{ precedes } receiving \end{aligned}$$

The axiom does not assert (and the theory has no vocabulary to say) that the receiving of the message was the result of this sending, only that at least one sending precedes the receiving.

The sendIn and receiveIn relations. The relations $send$ and $receive$ apply to terms that comprise the entire message being sent. It is also useful to discuss sending and receiving of terms that are included properly inside the message; this is done with the relations $sendIn$ and $receiveIn$.

The relation

$sendIn(action, agent, term)$

holds if $action$ is the sending by $agent$ of a message that includes $term$. Similarly, the relation

$receiveIn(action, agent, term)$

holds if $action$ is a receiving by $agent$ of a message that includes $term$.

The following axiom defines the relation $sendIn$.

$$\begin{aligned} & \forall (sending, sender, t0) \\ & \quad sendIn(sending, sender, t0) \\ & \Leftrightarrow \exists (t1) \quad send(sending, sender, t1) \wedge t0 \text{ inside } t1 \end{aligned}$$

A similar axiom defines the relation $receiveIn$. We can then establish a theorem that is analogous to the axiom Rcv .

$$\begin{aligned} & \forall (receiving, receiver, t) \\ & \quad receiveIn(receiving, receiver, t) \\ & \Rightarrow \exists (sending, sender, t) \\ & \quad sendIn(sending, sender, t) \wedge sending \text{ precedes } receiving. \end{aligned}$$

Henceforth, when we talk about a sending or receiving of a term, we shall actually mean the sending or receiving of any message that includes that term.

minSending and firstSending. An important concept for us will be the earliest sending of a term. Recall that the precedes relation is not total; this means we can distinguish between a first sending of the message, one which precedes all others, and a minimal sending, one which is not preceded by any other. Two sendings of the same term which are not temporally ordered may both be minimal, but neither will be first.

Here is the definition of *firstSendIn*:

$$\begin{aligned} & \forall(\textit{sending}, \textit{sender}, t) \\ & \quad \textit{firstSendIn}(\textit{sending}, \textit{sender}, t) \\ & \Leftrightarrow \textit{sendIn}(\textit{sending}, \textit{sender}, t) \\ & \quad \wedge (\forall(\textit{sending1}, \textit{sender1}) \\ & \quad \quad \textit{sendIn}(\textit{sending1}, \textit{sender1}, t) \\ & \quad \quad \Rightarrow (\textit{sending} \textit{precedes} \textit{sending1} \vee \textit{sending} = \textit{sending0})) \end{aligned}$$

The definition of *minSendIn* is analogous:

$$\begin{aligned} & \forall(\textit{sending}, \textit{sender}, t) \\ & \quad \textit{minSendIn}(\textit{sending}, \textit{sender}, t) \\ & \Leftrightarrow \textit{sendIn}(\textit{sending}, \textit{sender}, t) \\ & \quad \wedge (\forall(\textit{sending1}, \textit{sender1}) \\ & \quad \quad \textit{sendIn}(\textit{sending1}, \textit{sender1}, t) \Rightarrow \neg(\textit{sending1} \textit{precedes} \textit{sending})). \end{aligned}$$

It follows that a first sending will also be minimal, but not necessarily vice versa.

A basic property of minimal sendings is that any sending of a term is either a minimal sending itself or preceded by a minimal sending.

$$\begin{aligned} & \forall(\textit{sending}, \textit{sender}, t) \\ & \quad \textit{sendIn}(\textit{sending}, \textit{sender}, t) \\ & \Rightarrow \exists(\textit{sending0}, \textit{sender0}) \\ & \quad \textit{minSendIn}(\textit{sending0}, \textit{sender0}, t) \\ & \quad \wedge (\textit{sending0} \textit{precedes} \textit{sending} \vee \textit{sending0} = \textit{sending}). \end{aligned}$$

This follows directly from the well-foundedness of the precedes relation; rather than treating well-foundedness, we take it as an axiom in the theory.

It follows that any receiving of a term is preceded (strictly) by a minimal sending of that term, because any receiving is preceded by a sending:

$$\begin{aligned} & \forall(\textit{sending}, \textit{sender}, t) \\ & \quad \textit{receiveIn}(\textit{receiving}, \textit{receiver}, t) \\ & \Rightarrow \exists(\textit{sending0}, \textit{sender0}) \\ & \quad \textit{minSendIn}(\textit{sending0}, \textit{sender0}, t) \wedge \textit{sending0} \textit{precedes} \textit{receiving}. \end{aligned}$$

4.4 Nonces

Nonces are special terms that have a unique origin. Each nonce that is sent or received has a creator and a unique first sending. This is not taken to be true

of other terms; a phrase such as “now is the time for all good men” may have come from many senders, none of them first.

Nonce is declared to be the sort of all terms t that satisfy a relation $nonce?(t)$.

We define an agent $creator(nonce)$ and an action

$$firstSending(sending, sender, nonce)$$

by the following axiom:

$$\begin{aligned} & \forall(sending, sender, nonce) \\ & \quad sendIn(sending, sender, nonce) \\ & \Rightarrow firstSendIn(firstSending(sending, sender, nonce), creator(nonce), nonce). \end{aligned}$$

In other words, if a nonce is sent, it has a first sending by its creator. Furthermore, if a nonce is sent by anyone other than its creator, that agent must have previously received the nonce subsequent to its first sending.

$$\begin{aligned} & \forall(sending, sender, nonce) \\ & \quad sendIn(sending, sender, nonce) \\ & \quad \wedge sender \neq creator(nonce) \\ & \Rightarrow \exists(receiving) \\ & \quad \quad receiveIn(receiving, sender, nonce) \\ & \quad \quad \wedge receiving \text{ precedes } sending \\ & \quad \quad \wedge firstSending(sending, sender, nonce) \text{ precedes } receiving \end{aligned}$$

From these axioms, it follows that any sending of a nonce that is not itself a first sending is preceded by a first sending; also any receiving of a nonce is preceded by its first sending.

4.5 Keyed Hashing

A keyed hashing of a term is a summary of that term that is marked cryptographically by either of two principals in such a way that either of them can produce such a hash but no one else can. Furthermore, it is observable by either of the principals that one of them has produced and sent it. While it is not assumed that the term itself is recoverable from the hash, we do regard the term as being inside its hash. Furthermore, it is assumed (idealistically) that no other term will yield the same hash.

These facts are expressed in the following axioms:

$$\begin{aligned} & \forall(sending, sender, agentA, agentB, term) \\ & \quad minSendIn(sending, sender, hash(agentA, agentB, term)) \\ & \Rightarrow (sender = agentA \vee sender = agentB) \end{aligned}$$

In other words, a minimal sender of a hashed term must be one of its two principals. This is not true of every sender; after all, anyone can receive a hashed term and resend it.

$$\forall(agentA, agentB, t) \quad t \text{ inside } hash(agentA, agentB, t)$$

An additional axiom asserts that the hash function is one-to-one in all three of its arguments.

The theory we have outlined is sufficient to establish the authentication properties of the CRHH protocol.

5 Example Authenticity Proof

In this section we present the three stages of proof that together show mutual authenticity of the CRHH protocol. We first describe Alice’s authenticity obligation and its informal proof. We next describe how obligations such as this are generated automatically by PDA. Then we describe the incremental proof process, starting with the abstract CR protocol, for which proving one-way authenticity is trivial. The CR protocol is instantiated to CRH, by using identity for the challenge function and a hash function as the response function. The authenticity obligation for CRH follows from proving that this instantiation gives an interpretation. Finally, we use the theorem corresponding to the authenticity obligation for CRH to simplify the proof of the mutual authentication protocol CRHH.

5.1 The CRHH Protocol

In this protocol, *agentA* (Alice) and *agentB* (Bob) establish that they are in contact with each other. The protocol is illustrated in Figure 4.

Alice creates and sends a new nonce, *nonceA*:

$$\begin{aligned} & \textit{agentA} = \textit{creator}(\textit{nonceA}) \\ & \wedge \textit{firstSendIn}(\textit{sendingM}, \textit{agentA}, \textit{nonceA}). \end{aligned}$$

Bob receives the nonce *nonceA*:

$$\textit{receiveIn}(\textit{receivingM}, \textit{agentB}, \textit{nonceA}).$$

Bob then generates his own nonce, *nonceB*. He hashes the pair of the two nonces, and sends a larger pair that includes his nonce and the hashed pair:

$$\begin{aligned} & \textit{agentB} = \textit{creator}(\textit{nonceB}) \\ & \wedge \textit{firstSendIn}(\textit{sendingPair}, \textit{agentB}, \\ & \quad \langle \textit{nonceB}, \\ & \quad \textit{hash}(\textit{agentB}, \textit{agentA}, \langle \textit{nonceA}, \textit{nonceB} \rangle) \rangle) \end{aligned}$$

Alice receives this larger pair:

$$\begin{aligned} & \textit{receiveIn}(\textit{receivingPair}, \textit{agentA}, \\ & \quad \langle \textit{nonceB}, \\ & \quad \textit{hash}(\textit{agentB}, \textit{agentA}, \langle \textit{nonceA}, \textit{nonceB} \rangle) \rangle) \end{aligned}$$

She then hashes Bob’s nonce and resends it:

$$\textit{sendIn}(\textit{sendingN}, \textit{agentA}, \textit{hash}(\textit{agentA}, \textit{agentB}, \textit{nonceB})).$$

Bob receives his hashed nonce:

$$\textit{receiveIn}(\textit{receivingN}, \textit{agentB}, \textit{hash}(\textit{agentA}, \textit{agentB}, \textit{nonceB})).$$

We assume that these events occur in the given order.

5.2 Authentication Conclusions

Each of the above facts is apparent to one of the principals but not the other. For instance, Alice initially knows that she has created and sent her nonce, *nonceA*, but not that Bob has received it. Similarly, Bob knows that he has received a message, but not that it includes a nonce created by Alice. However,

based on what they observe, their knowledge of the properties of messages, nonces, and hashing, and some assumptions about the behavior of each other (and themselves) as participants in this protocol, they will be able to draw a number of conclusions.

Alice will be able to conclude that Bob has received her nonce; i.e., for some receivingB,

$$receiveIn(receivingB, agentB, nonceA).$$

Also, Alice will be able to conclude that $nonceB$ is a nonce, that Bob has created it, and that Bob is the first sender of the larger pair:

$$\begin{aligned} & nonce?(nonceB) \\ & \wedge agentB = creator(nonceB) \\ & \wedge firstSendIn(sendingPair, agentB, \\ & \quad \langle nonceB, \\ & \quad \quad hash(agentB, agentA, \langle nonceA, nonceB \rangle) \rangle). \end{aligned}$$

Also, Alice will be able to conclude that the events have occurred in the expected order; e.g, Bob has received her nonce after she has sent it.

Similarly, Bob will be able to conclude that $nonceA$ is a nonce that was created and first sent by Alice:

$$\begin{aligned} & nonce?(nonceA) \\ & \wedge agentA = creator(nonceA) \\ & \wedge firstSendIn(sendingA, agentA, nonceA). \end{aligned}$$

Also, Bob will be able to conclude that Alice has received his larger pair. Again, Bob will be able to deduce that the sequence of events has occurred in the expected order.

5.3 Honesty Assumptions

The reasoning that establishes these events depends on a number of so-called honesty assumptions, which assert that the two principals engaging in this protocol will behave in the expected manner. These conditions are not axioms; they are subformulas of the authentication conjectures that are proof obligations for the derivation of the protocol. Their variables are given meaning in those conjectures.

Alice's reasoning will depend on the assumption that Alice herself will not impersonate Bob and send out the same hashing of the pair of nonces,

$$hash(agentB, agentA, \langle nonceA, nonceB \rangle)$$

that she expects Bob to send out. This is expressed by the condition

$$\begin{aligned} & \neg(\exists(sillySending) \\ & \quad minSendIn(sillySending, agentA, \\ & \quad \quad hash(agentB, agentA, \langle nonceA, nonceB \rangle))). \end{aligned}$$

This might be called a rationality assumption rather than an honesty assumption, because if she does send it out and later receives the same term, she will have no way of knowing that Bob has received it; anyone could have sent it. (We need only forbid minimal sendings, because once the hash has been re-

leased there is no harm in sending it again.) The condition seems so obvious that we were unable to prove earlier formulations of Alice's authentication because we neglected to mention it. We had included Alice's assumptions about Bob's behavior but not her own.

Another assumption Alice must make is that Bob will not initially send out the hashed pair of nonces except in the context of his first sending of the larger pair. This is expressed by the condition

$$\begin{aligned} & \forall(\mathit{honestSending}) \\ & \quad \mathit{minSendIn}(\mathit{honestSending}, \mathit{agentB}, \\ & \quad \quad \mathit{hash}(\mathit{agentB}, \mathit{agentA}, \langle \mathit{nonceA}, \mathit{nonceB} \rangle)) \\ \Rightarrow & \quad \mathit{nonce}?(\mathit{nonceB}) \\ & \quad \wedge \mathit{agentB} = \mathit{creator}(\mathit{nonceB}) \\ & \quad \wedge \mathit{firstSendIn}(\mathit{honestSending}, \mathit{agentB}, \\ & \quad \quad \langle \mathit{nonceB}, \\ & \quad \quad \quad \mathit{hash}(\mathit{agentB}, \mathit{agentA}, \langle \mathit{nonceA}, \mathit{nonceB} \rangle) \rangle). \end{aligned}$$

We shall call this the appropriateness condition.

5.4 Alice's Reasoning

Let us reproduce the informal reasoning that Alice performs to conclude that Bob is the first sender of the larger pair, i.e.,

$$\begin{aligned} & \mathit{firstSendIn}(\mathit{sendingPair}, \mathit{agentB}, \\ & \quad \langle \mathit{nonceB}, \\ & \quad \quad \mathit{hash}(\mathit{agentB}, \mathit{agentA}, \langle \mathit{nonceA}, \mathit{nonceB} \rangle) \rangle). \end{aligned}$$

By hypothesis, we know that Alice has received the larger pair, i.e.,

$$\begin{aligned} & \mathit{receiveIn}(\mathit{receivingPair}, \mathit{agentA}, \\ & \quad \langle \mathit{nonceB}, \\ & \quad \quad \mathit{hash}(\mathit{agentB}, \mathit{agentA}, \langle \mathit{nonceA}, \mathit{nonceB} \rangle) \rangle) \end{aligned}$$

Hence, by the definition of $\mathit{receiveIn}$, because the components of a pair are inside the pair, Alice has received the second component of the pair, i.e.,

$$\begin{aligned} & \mathit{receiveIn}(\mathit{receivingPair}, \mathit{agentA}, \\ & \quad \mathit{hash}(\mathit{agentB}, \mathit{agentA}, \langle \mathit{nonceA}, \mathit{nonceB} \rangle)). \end{aligned}$$

Recall that any receiving of a term is preceded by a minimal sending of that term. Consequently, there are an action and an agent, called $\mathit{sendingPair}$ and $\mathit{senderPair}$, respectively, such that

$$\begin{aligned} & \mathit{minSendIn}(\mathit{sendingPair}, \mathit{senderPair}, \\ & \quad \mathit{hash}(\mathit{agentB}, \mathit{agentA}, \langle \mathit{nonceA}, \mathit{nonceB} \rangle)). \end{aligned}$$

Because the minimal sender of a hashed term must be one of its two principals, this means that

$$\mathit{senderPair} = \mathit{agentB} \vee \mathit{senderPair} = \mathit{agentA}.$$

By the rationality assumption for Alice, we know that Alice would not herself send out the hash she expects Bob to send as confirmation. Hence

$$\mathit{senderPair} = \mathit{agentB},$$

and

$$\text{minSendIn}(\text{sendingPair}, \text{agentB}, \text{hash}(\text{agentB}, \text{agentA}, \langle \text{nonceA}, \text{nonceB} \rangle)).$$

But now, by the appropriateness condition, Bob would not send out the hashed pair of nonces except in the context of the first sending of the larger pair. That is,

$$\text{firstSendIn}(\text{sendingPair}, \text{agentB}, \langle \text{nonceB}, \text{hash}(\text{agentB}, \text{agentA}, \langle \text{nonceA}, \text{nonceB} \rangle) \rangle),$$

which is part of the authentication conclusion for Alice.

This is an informal rendition of part of the formal proof found by SNARK; other parts of the proof establish that Bob has received Alice's nonce, *nonceA*, and that the required actions of the protocol have occurred in the expected order. The full, automatically constructed proof may be viewed at <http://www.kestrel.edu/software/pda/APWW06/>.

5.5 Automatic Generation of Authentication Obligations

In the previous sections, we have described how the facts concerning the CRHH protocol are derived from its desired run as specified in its PDA protocol diagram, and how they are divided into what each of the agents knows and what they must prove to ensure authentication. In general, the authentication obligations follow easily from the PDA specification of the desired run for a protocol, so we have automated their production. For each agent we generate a spec of what the agent knows from its own actions, and an authentication obligation that posits the existence of actions by the other agents performing roles in the protocol.

Generation of Knowledge Specs At the end of a protocol run, the agent knows the sends and receives it has performed and their order as specified in the protocol diagram. The spec for this knowledge consists of an action identifier for each send and receive and, for each send and receive, an axiom describing it, and an axiom for each nonce created by the agent describing that fact.

The axiom for a send by agent *agent* of *term* whose action identifier is *sending* is

$$\text{sendIn}(\text{sending}, \text{agent}, \text{term})$$

except when the term includes a newly-generated nonce, where we generate the stronger axiom

$$\text{firstSendIn}(\text{sending}, \text{agent}, \text{term}).$$

Note that this use of a stronger axiom does not mean we assume that the agent only sends *term* once, but we know that there must be a first sending, and it follows that any receiving of *term* must be preceded by this first sending.

The axiom for a receive by agent *agent* of *term* whose action identifier is *receiving* is

$$\text{receiveIn}(\text{receiving}, \text{agent}, \text{term})$$

Finally, the knowledge spec contains precedence statements for the action identifiers which are read from the precedence arrows in the protocol diagram.

The knowledge spec for Alice in CRHH is

op $x0$: *Nonce*
op $x1$: *Nonce*
op $ASendingN$: *Action*
op $AReceivingP$: *Action*
op $ASendingH$: *Action*
axiom
 $agentA = creator(x0)$
 $\wedge firstSendIn(ASendingN, agentA, x0)$
 $\wedge receiveIn(AReceivingP, agentA, \langle x1, hash(agentB, agentA, \langle x0, x1 \rangle) \rangle)$
 $\wedge send(ASendingH, agentA, hash(agentA, agentB, x1))$
 $\wedge ASendingN \text{ precedes } AReceivingP$
 $\wedge AReceivingP \text{ precedes } ASendingH.$

Generating Obligations The authenticity obligation for an agent is essentially that the actions of the other agents in the desired run of the protocol can be inferred. This ideal has to be relaxed in practice because of the nature of distributed computing. An important property of the logic presented above is that we can only infer prior actions. For example, if the last action of an agent is to send a message, then it is impossible for the agent to infer that the message was received. Therefore, the authenticity obligation we generate for an agent only considers actions prior to its last receive action.

The form of the generated obligation is an existential statement with variables for actions of the other agents, and a conjunction of the required properties. The form of the conjunct generated for each action is in most cases the same as that generated for the knowledge spec. The algorithm for generating the obligation starts from the last receive of an agent, following temporal links backwards in the diagram until there are no more. For each send or receive action visited, if it is by a different agent, an existential variable is created for it and an obligation conjunct added of the same form as for the knowledge spec. In addition, precedes obligations for each pair of actions are collected.

The authenticity obligation for Alice in CRHH is

$\exists(BReceivingC, BSendingP)$
 $receiveIn(BReceivingC, agentB, x0)$
 $\wedge sendIn(BSendingP, agentB, \langle x1, hash(agentB, agentA, \langle x0, x1 \rangle) \rangle)$
 $\wedge ASendingN \text{ precedes } BReceivingC$
 $\wedge BReceivingC \text{ precedes } BSendingP$
 $\wedge BSendingP \text{ precedes } AReceivingP.$

Generating Honesty Assumptions Some of the honesty assumptions can be generated automatically. Not all authentication protocols require honesty assumptions so generation is a user-specified option, and some honesty assumptions are domain-specific so there is an option for the user to supply them explicitly.

Honesty of the other agent means that the agent is following the protocol properly. What this amounts to is that, if we can determine that the other agent performed one of the actions in the protocol we can assume earlier ones were performed in the specified order. In particular, if the last send of the other agent can be determined to have occurred, then we can assume the earlier actions were performed. This condition turns out to be too stringent in practice, because, unless this last message sent is very simple, we can't infer that it was sent in the form specified in the desired run. For example, if the message is a pair, then even though we receive the pair, it is always possible that the other agent sent the two elements of the pair separately and a third agent bundled them together. Therefore, the condition for the honesty action is that the other agent sent one or both of the elements of the pair. We use a heuristic to avoid elements that we could not infer came from the other agent, but we allow for the user to override the heuristic choice.

In the CRHH example above, Bob's last send is

$$\text{sendIn}(\text{honestSending}, \text{agentB}, \\ \langle \text{nonceB}, \text{hash}(\text{agentB}, \text{agentA}, \langle \text{nonceA}, \text{nonceB} \rangle) \rangle).$$

The first element of the pair is a nonce that Alice cannot determine was sent by Bob without using the second element of the pair, so the sending of the second element is chosen as the condition. Moreover, if it is not the first send of this term, then anyone could have sent it, so the full condition generated is:

$$\begin{aligned} & \forall(\text{honestSending}) \\ & \quad \text{firstSendIn}(\text{honestSending}, \text{agentB}, \\ & \quad \quad \text{hash}(\text{agentB}, \text{agentA}, \langle \text{nonceA}, \text{nonceB} \rangle)) \\ \Rightarrow & \exists(\text{receivingB}) \\ & \quad \text{receiveIn}(\text{receivingB}, \text{agentB}, \text{nonceA}) \\ & \quad \wedge \text{receivingB precedes honestSending} \\ & \quad \wedge \text{nonce?}(\text{nonceB}) \\ & \quad \wedge \text{agentB} = \text{creator}(\text{nonceB}) \\ & \quad \wedge \text{firstSendIn}(\text{honestSending}, \text{agentB}, \\ & \quad \quad \langle \text{nonceB}, \text{hash}(\text{agentB}, \text{agentA}, \langle \text{nonceA}, \text{nonceB} \rangle) \rangle) . \end{aligned}$$

This is actually stronger than the hand-generated appropriateness condition used for the proof of the earlier section.

We have not yet attempted to generate the rationality condition automatically. Generating this condition requires reasoning that Alice should not reveal secrets prematurely.

5.6 The Abstract CR Protocol

In the basic CR protocol, *agentA* (Alice) authenticates *agentB* (Bob). The protocol is illustrated at the top of Figure 4. It has abstract functions *challenge* (*c*) and *response* (*r*) that satisfy the necessary condition for Alice to authenticate Bob.

Alice creates a new nonce, *nonceA*, so we know

$$\text{agentA} = \text{creator}(\text{nonceA}),$$

and sends a challenge

$$\text{challenge}(\text{agent}A, \text{agent}B, \text{nonce}A).$$

In the abstract theory, we do not specify what that challenge is. Thus we have an action $\text{sending}C$ such that

$$\text{minSendIn}(\text{sending}C, \text{agent}A, \text{challenge}(\text{agent}A, \text{agent}B, \text{nonce}A)).$$

Alice then receives the corresponding response; i.e., there is an action $\text{receiving}R$ such that

$$\text{receiveIn}(\text{receiving}C, \text{agent}A, \text{response}(\text{agent}A, \text{agent}B, \text{nonce}A)).$$

We assume that these events occur in order, i.e.,

$$\text{sending}C \text{ precedes } \text{receiving}C.$$

We include an honesty assumption, called the *rationality assumption*, that Alice does not herself send out the response she expects from Bob; otherwise, she will have no way of determining if the response she receives comes from Bob or is a copy of her own message:

$$\neg \exists (\text{sillysending}) \\ \text{sendIn}(\text{sillysending}, \text{agent}A, \text{response}(\text{agent}A, \text{agent}B, \text{nonce}A)).$$

For simplicity, we assume Alice and Bob are distinct agents, i.e.,

$$\text{agent}A \neq \text{agent}B.$$

In fact, if we didn't assume this, we would be able to deduce it, because we will show that Bob sends the response and we assume, in the *rationality assumption*, that Alice does not; but the proofs are a bit shorter and clearer if we assume the agents are distinct.

The challenge and response functions must be such that, from the above assumptions, we are able to draw the following conclusions:

Bob has received the challenge, i.e., there is an action $\text{receiving}C$ such that

$$\text{receiveIn}(\text{receiving}C, \text{agent}B, \text{challenge}(\text{agent}A, \text{agent}B, \text{nonce}A)).$$

Bob has then sent out the corresponding response; i.e., there exists an action $\text{sending}C$ such that

$$\text{minSendIn}(\text{sending}R, \text{agent}B, \text{response}(\text{agent}A, \text{agent}B, \text{nonce}A)).$$

These events all occur in the appropriate order, i.e.

$$\text{sending}C \text{ precedes } \text{receiving}C \text{ precedes } \text{sending}R \text{ precedes } \text{receiving}R.$$

The expression of the above property as a logical sentence is called the *Challenge-Response axiom*.

5.7 Proof of an Image of the Abstract *Challenge-Response Axiom*

The protocol CRH is defined as an instance of the abstract protocol CR, so to show authenticity we need to prove the interpretation from the CR spec to the theory of hashing, in which the challenge function is simply the nonce itself and the response is the hash function:

$$\{\text{challenge}(\text{agent}A, \text{agent}B, \text{nonce}A) \mapsto \text{nonce}A,$$

$$response(agentA, agentB, nonceA) \mapsto hash(agentA, agentB, nonceA)\}.$$

Hence we assume the antecedents of the *Challenge-Response* axiom, instantiated under the interpretation, and prove that the corresponding consequents follow. We shall consider only the instance of the consequent

$$minSendIn(sendingR, agentB, response(agentA, agentB, nonceA));$$

that is, we want to prove

$$minSendIn(sendingR, agentB, hash(agentA, agentB, nonceA)),$$

for some action *sendingR*.

We know the instance of the antecedent, that Alice has received the hashed version of her nonce

$$receiveIn(receivingR, agentA, hash(agentA, agentB, nonceA)).$$

By a theorem of our basic theory of messages, any receiving of a message is preceded by a minimal sending of that message. Hence, there exists an earlier action *sendingR* and an agent *senderR* such that

$$minSendIn(sendingR, senderR, hash(agentA, agentB, nonceA)).$$

We know that a minimal sender of a hashed term must be one of its two principals; hence

$$senderR = agentA \vee senderR = agentB.$$

But, by the *rationality assumption*, Alice would not send out the same message she is expecting Bob to send as part of the authentication process; hence Alice is not the sender, and

$$senderR = agentB.$$

It follows that Bob is the minimal sender, i.e.,

$$minSendIn(sendingR, agentB, hash(agentA, agentB, nonceA)),$$

as we wanted to show.

This is just part of the proof. The full proof, as discovered by the theorem prover SNARK, appears in <http://www.kestrel.edu/software/pda/APWW06/> along with other proofs from this paper.

The protocol CRH.op is just the instance of CRH with Alice and Bob reversed, so that Bob can authenticate Alice.

5.8 Proving Mutual Authentication in CRHH

Our goal is to prove the above authenticity obligation for Alice using the authenticity theorem for CR that was generalized and instantiated in CRH to give:

$$\begin{aligned} & \forall(x: Nonce, sendingC, receivingR) \\ & \quad agentA = creator(x) \\ & \quad \wedge firstSendIn(sendingC, agentA, x) \\ & \quad \wedge receiveIn(receivingR, agentA, hash(agentB, agentA, x)) \\ & \quad \wedge sendingC \text{ precedes } receivingR \\ & \quad \wedge \neg \exists(sillysending) sendIn(sillysending, agentA, hash(agentB, agentA, x)) \\ & \Rightarrow \exists(receivingC, sendingR) \end{aligned}$$

$$\begin{aligned}
& \text{receiveIn}(\text{receivingC}, \text{agentB}, x) \\
& \wedge \text{sendIn}(\text{sendingR}, \text{agentB}, \text{hash}(\text{agentB}, \text{agentA}, x)) \\
& \wedge \text{sendingC precedes receivingC} \\
& \wedge \text{receivingC precedes sendingR} \\
& \wedge \text{sendingR precedes receivingR}
\end{aligned}$$

However, one of the preconditions of this theorem,

$$\text{receiveIn}(\text{receivingR}, \text{agentA}, \text{hash}(\text{agentB}, \text{agentA}, x))$$

does not match the knowledge spec fact

$$\text{receiveIn}(\text{AReceivingP}, \text{agentA}, \langle x1, \text{hash}(B, \text{agentA}, \langle x0, x1 \rangle) \rangle)$$

because the latter is hashing a pair of nonces rather than a single nonce. To make the proof go through we need to generalize the precondition to

$$\begin{aligned}
& \text{receiveIn}(\text{receivingR}, \text{agentA}, \text{hash}(\text{agentB}, \text{agentA}, y)) \\
& \wedge x \text{ inside } y.
\end{aligned}$$

This generalization step is currently done by hand and proved from the spec of CR. We could automate such generalizations but we do not have sufficient experience to know when it is desirable.

Proving the authenticity obligation for Bob is similar to that for Alice, although in this case the extra generalization step is unnecessary. Both proofs take less than a second. For comparison we also proved these obligations directly from the properties of *hash* and the proofs took about 20 and 10 seconds, respectively for Alice's and Bob's obligations.

5.9 Theorem Proving Issues

As we have mentioned, the proof obligations were established by the theorem prover SNARK, a general-purpose first-order theorem prover that employs machine-oriented inference rules, including resolution (for general reasoning) and paramodulation (for equality reasoning). Automatic theorem provers are entirely domain-independent, but may have difficulty finding complex proofs unless equipped with domain-specific strategic controls. To achieve reasonable running times, we introduced a number of measures, most prominently a strategic ordering on the symbols in the authentication theory. The effect of this ordering is to ensure that symbols higher in the ordering are replaced by lower symbols, rather than the reverse.

Commonly, when a symbol is defined in term of another, we declare it to be higher in the ordering, so that the axioms that define the symbol can be used to unfold the definition. For instance, because minimal sending is defined in terms of sending, we declare that *minSendIn* is higher than *sendIn*. Other orderings are determined experimentally with a test suite.

With or without the ordering, most of the conjectures are proved in fractions of a second. The ordering, however, has a dramatic effect on the time to prove the authentication proof obligations. Without the ordering, the authentication conjectures are not proved, even if we allow the theorem prover to run for ten minutes.

While automatic validation of this kind does not convey an absolute correctness guarantee, it can uncover bugs in a protocol and increase our confidence in its correctness. The authentication of the protocols depends on the correctness of the axioms in our theory. The validation of these axioms has been an informal and gradual process. We see whether conjectures we expected to be true are proved. When they are not, we correct the axiom or the conjecture, accordingly. Most commonly, failed proofs have been the fault of missing assumptions in the conjecture, not errors in axioms.

Periodically we have reasoned forward from the axioms in an attempt to detect inconsistencies. These proof searches are permitted to run overnight. So far, no inconsistencies have been detected. We do not attempt to verify consistency by mapping our theory into another that is believed to be consistent. That would be a labor-intensive effort of limited value: a consistent theory can still be wrong.

Comparison to other Authenticity Verification Work. The theorem proving component of this work bears comparison to some other work in the field, e.g., Paulson’s [33]. Our work was done using SNARK, a first-order, automatic theorem prover with no support for inductive reasoning; Paulson’s was done using Isabelle, a higher order, interactive system that relies strongly on induction. Some of our axioms could actually have been proved by well-founded induction over the *precedes* relation, if we were using an inductive theorem prover. Paulson used induction on traces rather than well-founded induction. He was able to develop tactics that automated much of the theorem-proving process, fortunately: some Isabelle commands expanded to thousands of proof steps.

By relying on the factoring of the theory, we were able to decompose the authentication proofs into lemmas, none of which required more than 100 proof steps, or 1 second, to discover. Once the settings and orderings for the theory are decided on, the proofs require no interaction at all. While the protocols we dealt with were relatively simple, we believe factoring the theory will allow the treatment of more complex protocols without straining the theorem prover. Also, we believe the use of an automatic theorem prover will allow the derivation of protocols by security researchers who are not interested in the proof-discovery process. On the other hand, we are working on interfaces to other theorem provers from Specware, including Isabelle, so that theorem-proving experts can use other systems to perform the authenticity proofs.

6 Conclusions and Future Work

The goal of this work, and the guiding principle in designing PDA, has been to contribute to a methodology for incremental reasoning about security. The main problem in security engineering is that it begins where static reasoning about the environment ends; an active, adversarial environment is its subject, hampering any attempt at a straightforward component-based approach.

PDS formalizes a framework for protocol derivations and incremental reasoning about security [11, 13, 30, 34]. PDA provides automated support. Scalable reasoning about computational systems requires computational support to manage complexity and scope. Scalable reasoning about *distributed* computational

systems also requires computational support for representation of and interaction with the system. PDA supports the user by automatically generating authentication obligations for protocols, allowing for automatic proving of theorems with prover parameters specialized for the authentication theory presented here, and propagating theorems through the protocol derivation structure.

Factoring of a reasoning task into incremental steps often makes the difference between a feasible and an unfeasible task. A direct proof of a complex property of a complex protocol may be unfeasible; factoring it as a composite proof of simpler properties of simpler protocols may make it feasible. In the current work we have shown how factoring the proof of authenticity in a mutual authenticity protocol reduces the proof time by an order of magnitude. We intend to apply the approach to larger protocol derivations and expect larger gains.

The factorization of proofs in PDA matches the derivation structure of protocols. For protocol instantiations our formulation gives us the factoring inherent in spec interpretations. For PDA rules, which allow for restructuring we have shown the utility of generalization of authenticity theorems. In future work we will look at exploiting the structure of the individual rules. In addition, we intend to extend the scope of the reasoning in PDA to include secrecy properties along the lines of our work in [11, 34].

References

1. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols. *Inf. Comput.*, 148(1):1–70, 1999.
2. Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptol.*, 15(2):103–127, 2002.
3. Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.
4. Matthias Anlauf and Dusko Pavlovic. On specification carrying software, its refinement and composition. In H. Ehrig, B.J. Krüger, and A. Ertas, editors, *Proceedings of IDPT 2002*. Society for Design and Process Science, 2002.
5. Matthias Anlauf and Dusko Pavlovic. The protocol derivation assistant, 2005. <http://www.kestrel.edu/software/pda>.
6. Matthias Anlauf, Dusko Pavlovic, and Asuman Suenbuel. Deriving secure network protocols for enterprise services architectures. In *Proceedings of the IEEE International Conference on Communications (ICC 2006), Istanbul*, Piscataway, NJ, USA, June 2006. IEEE Press.
7. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.C. Hem, O. Kouchnarenko, J. Mantovani, S. Mdersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Vigan, and L. Vigneron. The Avispa tool for the automated validation of internet security protocols and applications. In *CAV 2005: Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, New York, NY, USA, 2005. Springer-Verlag New York, Inc.
8. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication, from proceedings of the royal society, volume 426, number 1871, 1989. In William Stallings, editor, *Practical Cryptography for Data Internetworks*. IEEE Computer Society Press, 1996.
9. R. M. Burstall and J. A. Goguen. Putting theories together to make specifications. In *IJCAI5*, pages 1045–1058, Cambridge, MA, August 22–25, 1977. IJCAI.

10. Iliano Cervesato, Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 55–69, 1999.
11. Iliano Cervesato, Catherine Meadows, and Dusko Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In Joshua Guttman, editor, *Proceedings of the 18th IEEE Computer Security Foundations Workshop*, pages 48–61. IEEE, 2005.
12. Anupam Datta, Ante Derek, John Mitchell, and Dusko Pavlovic. Secure protocol composition. *E. Notes in Theor. Comp. Sci.*, page 27 pp, 2003.
13. Anupam Datta, Ante Derek, John Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security protocols. *J. of Comp. Security*, 13:423–482, 2005.
14. Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system for security protocols and its logical formalization. In Dennis Volpano, editor, *Proceedings of CSFW 2003*, pages 109–125. IEEE, 2003.
15. G. Denker, J. Millen, and H. Ruess. The CAPSL integrated protocol environment. Technical Report SRI-CSL-2000-02, SRI International, October 2000.
16. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
17. Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, 1983.
18. Nancy Durgin, John Mitchell, and Dusko Pavlovic. A compositional logic for proving security properties of protocols. *J. of Comp. Security*, 11(4):677–721, 2004.
19. Nancy Durgin, John C. Mitchell, and Dusko Pavlovic. A compositional logic for protocol correctness. In Steve Schneider, editor, *Proceedings of CSFW 2001*, pages 241–255. IEEE, 2001.
20. Eclipse-Team. Eclipse, 2005. <http://www.eclipse.org>.
21. F. Javier Thayer Fabrega, Jonathan Herzog, and Joshua Guttman. Strand spaces: What makes a security protocol correct? *Journal of Computer Security*, 7:191–230, 1999.
22. GEF-Team. The Graphical Editing Framework (GEF), 2005. <http://www.eclipse.org/projects/gef>.
23. C. A. R. Hoare. Programs are predicates. In *Proc. of a discussion meeting of the Royal Society of London on Mathematical logic and programming languages*, pages 141–155, Upper Saddle River, NJ, USA, 1985. Prentice-Hall, Inc.
24. C.A.R. Hoare. Assertions: A personal perspective. *IEEE Annals of the History of Computing*, 25(2):14–25, 2003.
25. Kestrel Institute. *Specware System and Documentation*, 2004. <http://www.specware.org/>.
26. Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.
27. Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.
28. Ueli Maurer. Information-theoretic cryptography. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 47–64. Springer-Verlag, August 1999.
29. Catherine Meadows. A model of computation for the nrl protocol analyzer. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 84–89, 1994.

30. Catherine Meadows and Dusko Pavlovic. Deriving, attacking and defending the GDOI protocol. In Peter Ryan, Pierangela Samarati, Dieter Gollmann, and Refik Molva, editors, *Proceedings of ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 53–72. Springer Verlag, 2004.
31. A. Menezes, M. Qu, and S. Vanstone. Some new key agreement protocols providing mutual implicit authentication. In *SAC '95: Proceedings of the Selected Areas in Cryptography*, pages 22–32, London, UK, 1995. Springer-Verlag.
32. J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Mur ϕ . In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 141–151, 1997.
33. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
34. Dusko Pavlovic and Catherine Meadows. Deriving secrecy properties in key establishment protocols. In Dieter Gollmann and Andrei Sabelfeld, editors, *Proceedings of ESORICS 2006*, Lecture Notes in Computer Science. Springer Verlag, 2006. to appear.
35. Dusko Pavlovic and Douglas R. Smith. Composition and refinement of behavioral specifications. In *Automated Software Engineering 2001. The Sixteenth International Conference on Automated Software Engineering*. IEEE, 2001.
36. Dusko Pavlovic and Douglas R. Smith. Guarded transitions in evolving specifications. In H. Kirchner and C. Ringeissen, editors, *Proceedings of AMAST 2002*, volume 2422 of *Lecture Notes in Computer Science*, pages 411–425. Springer Verlag, 2002.
37. Vaughan Pratt. Modelling concurrency with partial orders. *Internat. J. Parallel Programming*, 15:33–71, 1987.
38. Steve Schneider. Verifying authentication protocols in CSP. *IEEE Trans. Softw. Eng.*, 24(9):741–758, 1998.
39. Claude E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28:656–715, 1949.
40. M. Stickel, R. Waldinger, and V. Chaudhri. *A guide to SNARK*. SRI International, 2000. <http://www.ai.sri.com/snark/tutorial.html>.