Toward Automatic Synthesis of Security Protocols[*]

Hassen Saïdi
System Design Laboratory
SRI International
Menlo Park, CA 94025, USA
www.sdl.sri.com/people/saidi/
saidi@sdl.sri.com

Abstract

The use of cryptographic protocols that enforce a variety of security properties has become more and more important in every day's Internet transactions. The task of designing such protocols is tedious and error prone: many protocols that were believed to be correct for years have been shown to contain subtle errors. Moreover, it is difficult to design protocols that are adaptable to different constraints on their execution environment such as CPU power and bandwith. Security protocols can be built from simple communication primitives provided by standard protocols, and from cryptographic primitives. The rules of the protocol, can be derived from a high-level specification of what the protocol is designed to achieve, and under what hardware and software constraints it will be used. It has been shown that most of these specifications are best captured from a knowledge and belief-based viewpoint. In this work, we propose a technique allowing the automatic synthesis of security protocols that satisfy, by construction, their logical specification. Such specifications can be expressed using high-level communication primitives. Primitives have security and cryptographic attributes that include confidentiality, integrity, authentication, and nonrepudiation, and can be implemented using public and secret cryptography.

Introduction

The evolution of the Internet through the last three decades is an indication of its future evolution into an infrastructure for service delivery. This evolution requires building an entirely new class of protocols, networks, and infrastructures that will search, access, provide, and assemble services. In this highly dynamic environment, and with the increasing number of Web-enabled devices that operate under different environments and with different computation and communication capabilities, it will be hard to provide standards for many of the possible applications. We expect that program synthesis tools will play an important role in building bridges between applications and services, and generating services on the fly by assembling, adapting, and synthesizing basic components. Moreover, the synthesis approach provides greater confidence in the correctness of the generated applications which would otherwise have to be carefully designed, tested, and verified, which is a tedious and error prone task.

Cryptographic protocols are an important component in this new architecture, and are designed to provide a secure communication capability over an insecure network in order to protect business transactions, individual privacy, and national critical resources. Cryptographic communication protocols are the basis of security in many distributed systems, and it is therefore essential to ensure that these protocols function correctly and do not exhibit vulnerabilities that can be maliciously exploited. In every day's Internet transactions, secure communication achieving authentication, fair exchange, nonrepudiation, and contract signing, protocols must be carefully designed, tested and verified. However, for new application, the task of designing such protocols is tedious and error prone. Protocols may operate under different CPU and bandwidth constraints. Moreover, protocols must be adaptable to changes in security policies such as the length of encryption keys or a limit on the lifetime of a certificate. It is therefore difficult to design such protocols to be adaptable to these different constraints.

Security protocols can be built from simple communication primitives provided by standard communication protocols, and from cryptographic primitives. The rules of the protocol, that is, message content, can be derived from a high-level specification of what the protocol is designed to achieve. We believe that when such a higher-level specification is expressed in a logical formalism to which one can associate a reasoning framework, it is possible to automatically synthesize security protocols from that specification. Figure 1 shows a description of the synthesis methodology. First, we define the protocol by a set of goals it is designed to achieve. A set of constraints on the environment can also be provided. Such constraints may be used to select the best protocol that satisfies the constraints among a set of possible protocols that satisfy the same goal. Once a high-level description of the protocol is given and a set of protocol rules is generated, one can compile it into an executable form, using standard communication primitives and standard encryption mechanisms.

[*]This research was partially supported by DARPA/AFRL contract F30602-00-C-0087

The constraints on the execution environment of the protocol might be used to generate a protocol that satisfies its functional specification but also takes advantage of the environments. For instance, if the device that runs the protocol has access to a powerful-enough CPU, but limited bandwidth, one can generate a protocol that uses short messages for communication, but heavily uses encryption that can be handled by the CPU power.

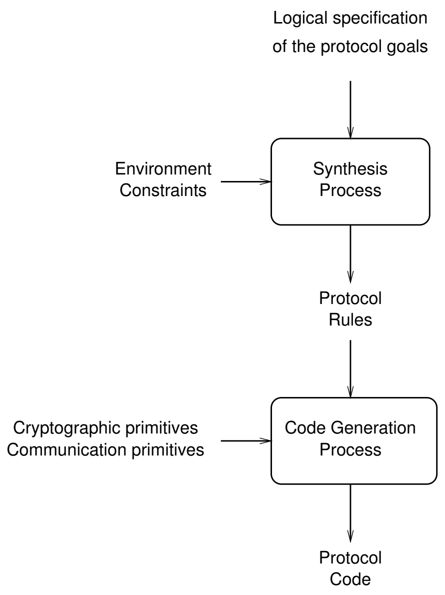

Figure 1: Protocol Synthesis

In this work, we propose a synthesis methodology that allows the automatic derivation of cryptographic protocols from a higher-level specification expressed in a logical form. Our synthesis methodology is based on derivation rules and consists of three main ingredients:

- A logic that describes assumptions on the environment, the initial configuration of the protocol, and the set of goals, that is, the logical specification of what the protocol is designed to achieve
- A proof system associated to the logic in order to decompose the set of goals into elementary goals that must be satisfied by the designed protocol
- A *realization* function that maps an elementary goal into protocol actions. That is, a function that maps the logical reasoning steps into protocol actions
- A set of constraints on the execution environment of the protocol

Middleware Architecture for Protocol Synthesis

Our synthesis methodology is part of a middleware architecture that facilitates the interoperability of web-enabled devices. The architecture (Denker & Saïdi 2001) shown in Figure 2, enables different devices to use the middleware capabilities and interfaces to request customized services that will become part of their applications. Devices are hardware and software components that have an interface to the middleware. The embedded systems may interact using the middleware to communicate and exchange information about their interfaces, constraints, and policies under which they are allowed to operate. Such information is expressed and exchanged in a common representation defined as part of an interface ontology for security services and protocols. The result of the interaction is code generated for the desired security services. The code generation is facilitated by a deductive system that implements a translation from high-level security service descriptions and system restrictions into logical properties, a proof system to reason about those properties, and a realization and synthesis module that generates code for the given goals and constraints. The synthesis module has the power to compute the necessary security protocols and will distribute the results to the devices, where the code is integrated into the application layer.

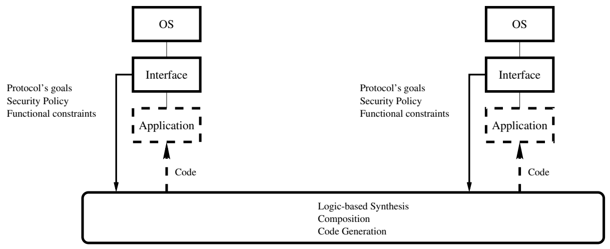

Figure 2: Middleware Architecture for Protocol Synthesis

Logic for Cryptographic Protocols and Synthesis Approaches

Parties involved in a protocol are considered as principals that have the ability to act when certain constraints are satisfied. Such constraints usually represent the state of knowledge of the principals. In such an application there is a strong link between knowledge and action. It has been shown (Halpern & Zuck 1992; Fagin *et al.* 1995) that a knowledge-based viewpoint gives a unifying framework for understanding, verifying, and designing protocols.

Logic of knowledge is the right level of abstraction for reasoning about security protocols. In security protocols, details about the data transfer protocol details are often omitted, and the focus is only on the cryptographic part that is best captured by the notion of knowledge. For instance, to deduce the correctness of an authentication protocol, one would need to prove that both parties are who they say they are. In a particular logic system, this could be X believes Y believes X and Y believes X believes Y, which is the formal way of saying that X and Y trust each other.

The use of logical analysis in protocol design is not widely implemented in the security protocol sector. This is mainly due to the lack of tools to implement such a logic system easily and also to the lack of trust in these well-known logic systems. This lack of trust may have a basis since flaws in logic systems have been uncovered. In addition, no one logic system has been proven to be both sound and complete. Logics like BAN (Burrows, Abadi, & Needham 1990) and GNY (Gong, Needham, & Yahalom 1990) have the most exposure as security logic systems and are the subject of much research. Syverson and van Oorschot also proposed a formal framework to unify several different cryptographic protocol logics (Syverson & van Oorschot 1994), and Abadi and Tuttle proposed a similar logic system (Abadi & Tuttle 1991b). However, most of the work on logics for security protocols is dedicated to authentication properties.

The two main techniques for the automatic derivation of protocols, are the model-based and the proof-based techniques. Recently, a model-based technique for the automatic generation of authentication protocols has been proposed by Perrig and Song (Perrig & Song 2000b; 2000a). Their technique is implemented as a procedure that takes as input a specification of the security properties that the protocol must satisfy, and a system requirement that includes a metric corresponding to the cost or overhead of the protocol. The metric imposes a limit on message size. During the protocol generation, all possible protocols up to a maximum cost threshold are generated. A model-checker (Song 1999) is used as a protocol screener to verify the generated protocols in order to eliminate those that do not meet the security specification. The logic that expresses the security properties of the protocols is weak in the sense that different protocols can be generated from the same set of properties, showing that the logic does not capture all the differences that exist between the generated protocols although they satisfy the same set of goals.

Different proof-based synthesis techniques have been proposed recently (Buttyán, Staamann, & Wilhelm 1998; Monniaux 1999; Clark & Jacob 2000). In (Buttyán, Staamann, & Wilhelm 1998) it was proposed to extend a BAN-like logic with a synthetic rules that when reversed can be used in a systematic way to design protocols. The synthetic rules introduce an abstract notion of channels, and define the set of readers and writers for a channel.

We believe that a logic-based synthesis methodology is more suitable for security protocols. The usefulness of a complete, correct logic system is quite substantial. The availability of a logic suitable for reasoning about security protocols greatly simplifies the protocol designer's task. Another useful side effect from using logic systems is that one is required to explicitly formalize the assumptions one makes, and this forces the use of a more stringent design methodology. Using a stringent methodology can only increase confidence in the resultant protocol.

Our logical framework is a general framework where authentication, confidentiality, fair exchange, nonrepudiation, contract signing, and secure group communication properties can be expressed. Our framework captures propositional attitudes as knowledge, belief, trust, and different notions of evidence. We use the BAN logic as a basis for the definition of our logic.

Figure 3 shows the formulas of the BAN logic and their informal meaning. The semantics of the belief modality is similar to the one of knowledge and is expressed by the axiom: $P| \equiv X \;\Rightarrow\; \vdash X$. We use the following belief production:

$Formula$	$Meaning$
$P\mid\equiv X$	P believes X
$P \triangleleft X$	P sees X
$P\mid\sim X$	P once said X
$P\mid\Rightarrow X$	P has jurisdiction over X
$\sharp(X)$	X is fresh
$P \overset{K}{\leftrightarrow} Q$	K is a symmetric key for P and Q
$P \overset{X}{\rightleftharpoons} Q$	P and Q share secret X
$\overset{+K}{\mapsto} Q$	$+K$ is the public key of Q
$\{X\}_K$	X encrypted with K
$\{X\}_{K^{-1}}$	X encrypted with the private key k^{-1} corresponding to the public key K
$\langle X\rangle_Y$	X combined with Y
(X, Y)	pair of X and Y

Figure 3: BAN logic formulas

Belief production rules

R4

$$\frac{P|\equiv \sharp(X) \quad P|\equiv Q|\sim X}{P|\equiv Q|\equiv X}$$

R5

$$\frac{P|\equiv Q|\Rightarrow X \quad P|\equiv Q|\equiv X}{P|\equiv X}$$

In our approach, we also use cryptographic channels that achieve some security goal. We define a more general and expressive notion of channels that have the following properties: integrity, confidentiality, authentication, and non-repudiation. Principals may use a combination of such channels, and may send channels to each other in the spirit of the SPI calculus (Abadi & Gordon 1999) through meta-channels.

Secure Communication Channels

Cryptography has four major goals that can be combined to provide most other security goals (Menezes, van Oorschot, & Vanstone 1996):

- *Confidentiality*, also called privacy or secrecy, is the oldest and most studied goal of cryptography. It assures that data can be read only by authorized persons. Many solutions using symmetric-key or public-key encryption have been proposed in literature.
- *Data integrity* prevents an unauthorized person from altering a message. *MDCs* (manipulation detection codes) based on cryptographic hash functions are generally used to provide such a service.
- *Authentication* has been studied extensively in the literature, and many discussions have arisen concerning its definition. Here we will consider two notions of authentication: message and entity. *Message authentication*, or data

origin authentication, provides the identity of the author of a message to a given recipient. Message authentication can be implemented using *secure envelopes* or *MACs* (message authentication codes) based on keyed cryptographic hash functions. *Entity authentication* provides an identification of an entity in a communication. An important difference between these two notions of authentication is that message authentication is not limited to a certain time period, while entity authentication is limited to the duration of the communication. Entity authentication needs the execution of a protocol. Such a protocol could, for instance, consist in sending some *fresh* information, while assuring message authentication.

- *Nonrepudiation* is the property that binds an entity to a message. A complete nonrepudiation service must ensure both *nonrepudiation of origin* and *nonrepudiation of receipt* (Zhou 1996). Nonrepudiation of origin provides *evidence* to the recipient of a message about the identity of the author who wrote the message. Until today, the only way of providing such evidence has been through using digital signatures. The difference is that nonrepudiation provides evidence that can be shown to an adjudicator about the identity of the author, while authentication only assures that the recipient is convinced of the identity of the author. Nonrepudiation of receipt provides an originator of a message with evidence that the recipient received a previously sent message. A complete nonrepudiation service cannot be implemented by a single cryptographic primitive. It needs a non-repudiation protocol to be run between two entities.

Communication channels are logical connections between principals. On addition to delivering messages, they can provide other cryptographic services. We define four channels that provide confidentiality, data integrity, message authentication, and nonrepudiation of origin.

Definition 1 (integrity channel) *An A-integrity channel assures that if entity A sends a message on the network, everyone will receive a nonaltered message.*

Definition 2 (authentic channel) *An AB-authentic channel provides message authentication between the two entities A and B: whenever a message arrives on an AB-authentic channel B, can be sure that the message was sent by A. Note that the message does not need to be fresh and that B can not convince anyone else that A is the author of the message. Moreover, authentication does not require the message to be secret.*

Definition 3 (nonrepudiable channel) *A non-repudiable channel provides nonrepudiation of origin. When a message arrives on an A-nonrepudiable channel, everyone is convinced that A is the author of the message.*

Definition 4 (confidential channel) *A B-confidential channel provides the service that only the author and B can read the content of message m.*

These channels can be used as building blocks to design more complex services. Channels can be implemented using a variety of cryptographic primitives. For instance, a confidential channel established between A and B can be implemented either by using the public key of B, or by using a shared secret key between A and B. It could even be implemented using a physically secure channel. Such a choice of implementation may depend on the availability of a public key infrastructure, or adequate software and hardware for generating shared keys. However, all of these possibilities guarantee the requirements of a B-confidential channel.

We can establish a hierarchy between these channel types. A nonrepudiable channel offers a stronger service than an authentic channel: as an A-nonrepudiable channel convinces everyone that A is the author of the message, it also convinces B in particular. In the same way, an A-integrity channel can be implemented by the stronger AB-authentic channel or an A-nonrepudiable channel, as changing the message m would change the author. A consequence of this hierarchy is that whenever we need a channel offering integrity services, we can implement it by an authentic or a nonrepudiable channel.

In practice, more complex services than those provided by the above-defined channels are needed. Therefore it is important to have the possibility to combine the services of the four basic channels. To do so, we define a message m that can be sent on a channel as either data or a channel in the spirit of the SPI calculus (Abadi & Gordon 1999) through meta-channels. Consider the following example, where A wants to send a message m to B that has first been digitally signed by A and then been encrypted for B:

$$\mathsf{send}_{B-c}(A,\ \mathsf{send}_{A-nr}(A, m,))$$

This example shows the expressive power of our model. B is the only one except the sender who can get the information sent on the confidential channel. Receiving a nonrepudiable channel means that B has all the knowledge to build this channel. B can thus decide to send this information to everyone else—and convince everyone else that A is the originator of message m—or decide to keep this information secret. By the same mechanism we can implement nested encryption and other more complex cryptographic mechanisms, while staying at a high level.

Computational Model

Our computational model is based on execution traces similar to the one proposed by Abadi and Tuttle (Abadi & Tuttle 1991a) for the BAN logic. We also use Paulson's approach (Paulson 1998; Millen & Ruess 2000) to model the behavior of principals, where protocols are inductively defined as sets of traces. A trace is a list of communication events. We consider protocols that are interactions between a set of principals using communication channels. A run of the protocol is a sequence σ of states. Each state corresponds to the execution of a communication step via a specified channel. We assume that all principals can read all messages. Let Q denote the global state space of the system. In a system state $q \in Q$, we denote by $trace(q)$ the set of messages that have been exchanged so far and by $\underline{trace}(q)$ the message contents that occur in $trace(q)$. Given a set of fields S, the following sets are used: $\mathsf{Parts}(S)$, $\mathsf{Analz}(S)$, and $\mathsf{Synth}(S)$. $\mathsf{Parts}(S)$ is the set of fields and subfields that occur in S. $\mathsf{Analz}(S)$ is the set of fields that can be extracted from elements of S without breaking the cryptosystem. $\mathsf{Synth}(S)$ is the set of fields that can be constructed from elements of S by concatenation and encryption. Formal definitions can be found in (Paulson 1998) or (Millen & Ruess 2000). In a state q, the set of fields that G can access is then

$$\mathsf{Know}(G, q) \quad = \quad \mathsf{Analz}(I(G) \cup \underline{trace}(q)).$$

This is the set of fields that G can obtain from its initial knowledge $I(G)$ and the messages seen so far. In the BAN logic, the semantics of the belief modality is defined as knowledge. Therefore, we use the trace model and the predicates $\mathsf{Parts}(S)$, $\mathsf{Analz}(S)$, and $\mathsf{Synth}(S)$ to provide a semantics for BAN logic. Thus, our model allows us to characterize the state where a certain formula is valid as a state in a particular sequence of events that corresponds to the messages observed so far. This allows us to construct, for a set of goals expressed in the BAN logic, a trace of events that starts from the state that satisfies the initial configuration of the protocol, and leads to the state where all goals are satisfied. Thus, in any state q, the belief of an agent A is characterized as the set $\mathsf{Know}(A, q)$

$q \models A| \equiv S \quad$ iff $\quad B \in \mathsf{Know}(A, q)$

Communication event $A \rightarrow \langle M \rangle_{mode} \rightarrow B$ expresses the fact that A sends a message M to B with the attribute $mode$ that can be confidential, nonrepudiable, and authenticated, and preserve the integrity of messages. The attributes can be considered as logical channels between A and B. Channels can be implemented using a variety of cryptographic primitives. For instance, a confidential channel established between A and B can be implemented either by using the public keys of A and B, or by using a shared secret key between A and B. Such choice of implementation may be decided depending on the availability of a public key infrastructure, or adequate software and hardware for generating shared keys.

In its simple form, a message M can consist of data. It can also consist of a belief formula. Thus, it is possible to express how A can transfer knowledge to B.

Example

As an example of the feasibility of our approach, we show how the well-known Needham-Schroeder authentication protocol can be generated from its logical specification, expressed in the BAN logic (Burrows, Abadi, & Needham 1990). Following is a complete derivation of the Needham-Schroeder protocol using BAN logic, in which initial and final states are, respectively

Initial State:

$S| \equiv \overset{+B}{\mapsto} B$
$S| \equiv \overset{+A}{\mapsto} A$
$A| \equiv S| \Rightarrow \overset{+B}{\mapsto} B$
$A| \equiv \sharp(N_a)$
$B| \equiv S| \Rightarrow \overset{+A}{\mapsto} A$
$B| \equiv \sharp(N_b)$

Goals:

$$\begin{array}{ll} & A| \equiv \overset{+B}{\mapsto} B \\ \wedge & B| \equiv \overset{+A}{\mapsto} A \\ \wedge & A| \equiv B| \equiv A \xleftrightarrow{N_b} B \\ \wedge & B| \equiv A| \equiv A \xleftrightarrow{N_a} B \end{array}$$

that can be used. The derivation process consists in using a set of rules that can correspond to proportional logic rules, or BAN logic rules. The derivation process terminates when all the goals are realized. From the derivation process, the communications are extracted and ordered in time. The resulting ordered communication messages are the derived protocol.

For synthesis purposes, we augment the BAN logic rules with additional production rules that allows the generation of protocol rules. The derivation process is presented in a deductive proof style where a combination of the BAN logic proof rules, propositional proof rules, and additional rules are used. A particular rule is added. This is the *realization* rule that allows us to build protocol rules from the beliefs of the principals. Also, a precedence rule is added to express chronological order between BAN formulas. For instance, one can assume that the formula $B| \equiv A \xleftrightarrow{N_b} B$ becomes valid *before* formula $A| \equiv B| \equiv A \xleftrightarrow{N_b} B$. A similar notion is used in the strand spaces model (Thayer, Herzog, & Guttman 1998). Thus, we use the following

precedence rule

$$\frac{B| \equiv A \xleftrightarrow{N_b} B \quad B \rightarrow \langle B| \equiv A \xleftrightarrow{N_b} B \rangle_s \rightarrow A}{A| \equiv B| \equiv A \xleftrightarrow{N_b} B}$$

that expresses the fact that A believes that B believes that B and A share a secret, B must first believe that it indeed shares the secret with A, and then it communicates this fact to A in a secure way expressed by $B \rightarrow \langle B| \equiv A \xleftrightarrow{N_b} B \rangle_s \rightarrow A$ that preserves the shared secret. This usually indicates that B acknowledges an earlier message containing the shared secret and sent by A.

$\Downarrow$

Goal: 1

$$\begin{array}{ll} & A| \equiv \overset{+B}{\mapsto} B \\ \wedge & B| \equiv \overset{+A}{\mapsto} A \\ \wedge & A| \equiv B| \equiv A \xleftrightarrow{N_b} B \\ \wedge & B| \equiv A| \equiv A \xleftrightarrow{N_a} B \end{array}$$

Rule? (`Split`)

$\Downarrow$

Goal: 1.1

$A \models \overset{+B}{\mapsto} B$

Rule? (apply **R5**)
$\Downarrow$

Goal: 1.1
$$(\exists Q : \texttt{Principal}) : \quad A \models Q \Rightarrow \overset{+B}{\mapsto} B \;\wedge\; A \models Q \models \overset{+B}{\mapsto} B$$

Rule? (inst Q with S)
$\Downarrow$

Goal: 1.1
$$A \models S \Rightarrow \overset{+B}{\mapsto} B \;\wedge\; A \models S \models \overset{+B}{\mapsto} B$$

Rule? (split)
$\Downarrow$

Goal: 1.1.1
$A \models S \Rightarrow \overset{+B}{\mapsto} B$

Rule? (assert)
$\Downarrow$

Goal: 1.1.2
$A \models S \models \overset{+B}{\mapsto} B$

Rule? (assert)
$Asserting$

$$A \models S \Rightarrow \overset{+B}{\mapsto} B \quad implies \quad A \models S \models \overset{+B}{\mapsto} B$$

$\Downarrow$

Goal: 1.2
$$(\exists Q : \texttt{Principal}) : \quad B \models Q \Rightarrow \overset{+A}{\mapsto} A \;\wedge\; B \models Q \models \overset{+A}{\mapsto} A$$

Rule? (inst Q with S)
$\Downarrow$

Goal: 1.2
$$B \models S \Rightarrow \overset{+A}{\mapsto} A \;\wedge\; B \models S \models \overset{+A}{\mapsto} A$$

Rule? (split)
$\Downarrow$

Goal: 1.2.1
$B \models S \Rightarrow \overset{+A}{\mapsto} A$

Rule? (assert)
$\Downarrow$

Goal: 1.2.2
$B \models S \models \overset{+A}{\mapsto} A$

Rule? (assert)
$Asserting$

$$B \models S \Rightarrow \overset{+A}{\mapsto} A \quad implies \quad B \models S \models \overset{+A}{\mapsto} A$$

$\Downarrow$

Goal: 1.3
$A \models B \models A \overset{N_b}{\longleftrightarrow} B$

Rule? (apply precedence rule)
$\Downarrow$

Goal: 1.3.1
$B \models A \overset{N_b}{\longleftrightarrow} B$

Rule? (realize)
$\Downarrow$

Goal: 1.3.1
$$B \models \sharp N_b \wedge B \to \langle N_b \rangle_s \to A \;\vee\; A \models \sharp N_b \wedge A \to \langle N_b \rangle_s \to B$$

Rule? (assert)
$\Downarrow$

Goal: 1.3.1
$B \models \sharp N_b \wedge B \to \langle N_b \rangle_s \to A$

Rule? (split)
$\Downarrow$

Goal: 1.3.1.1
$B \models \sharp N_b$

Rule? (assert)
$\Downarrow$

Goal: 1.3.1.2
$B \to \langle N_b \rangle_s \to A$

Rule? (realize)
$\Downarrow$

Goal: 1.3.1.2
$$(\exists K : \texttt{Key}) : \quad B \models \overset{+K}{\mapsto} A \;\vee\; B \models A \overset{K}{\longleftrightarrow} B \;\wedge\; B \to \langle \{N_b\}_K \rangle \to A$$

Rule? (inst K with A)
$\Downarrow$

Goal: 1.3.1.2
$$B \models \overset{+A}{\mapsto} A \;\vee\; B \models A \overset{A}{\longleftrightarrow} B \;\wedge\; B \to \langle \{N_b\}_A \rangle \to A$$

Rule? (assert)
$\Downarrow$

Goal: 1.3.1.2
$B \to \langle \{N_b\}_A \rangle \to A$

Rule? (realize)

$\Downarrow$

$Production$

Produce rule $B \longrightarrow A: \{N_b, B\}_A$

Goal: 1.3.2
$B \rightarrow \langle B| \equiv A \overset{N_b}{\longleftrightarrow} B\rangle_s \rightarrow A$

Rule? (acknowledge)
$\Downarrow$

$Production$

Produce rule $A \longrightarrow B: \{N_b, A\}_B$

Goal: 1.4
$B| \equiv A| \equiv A \overset{N_a}{\longleftrightarrow} B$

Rule? (apply precedence rule)
$\Downarrow$

Goal: 1.4.1
$A| \equiv A \overset{N_a}{\longleftrightarrow} B$

Rule? (realize)
$\Downarrow$

Goal: 1.4.1
$$\begin{array}{l} B| \equiv \sharp N_a \wedge B \rightarrow \langle N_a\rangle_s \rightarrow A \\ \vee\ A| \equiv \sharp N_a \wedge A \rightarrow \langle N_a\rangle_s \rightarrow B \end{array}$$
Rule? (assert)
$\Downarrow$

Goal: 1.4.1
$A| \equiv \sharp N_a \wedge A \rightarrow \langle N_a\rangle_s \rightarrow B$

Rule? (split)
$\Downarrow$

Goal: 1.4.1.1
$A| \equiv \sharp N_a$

Rule? (assert)
$\Downarrow$

Goal: 1.4.1.2
$A \rightarrow \langle N_a\rangle_s \rightarrow B$

Rule? (realize)
$\Downarrow$

$$(\exists K : \texttt{Key}) : \quad \begin{array}{ll} & A| \equiv \overset{+K}{\mapsto} B \ \vee\ A| \equiv A \overset{K}{\longleftrightarrow} B \\ \wedge & A \rightarrow \langle \{N_a\}_K\rangle \rightarrow B \end{array}$$
Rule? (inst K with B)
$\Downarrow$

Goal: 1.4.1.2
$$\begin{array}{ll} & A| \equiv \overset{+B}{\mapsto} B \ \vee\ A| \equiv A \overset{B}{\longleftrightarrow} B \\ \wedge & A \rightarrow \langle \{N_a\}_B\rangle \rightarrow B \end{array}$$
Rule? (assert)
$\Downarrow$

Goal: 1.4.1.2
$A \rightarrow \langle \{N_a\}_B\rangle \rightarrow B$

Rule? (realize)
$\Downarrow$

$Production$

Produce rule $A \longrightarrow B: \{N_a, A\}_B$

Goal: 1.4.2
$A \rightarrow \langle A| \equiv A \overset{N_a}{\longleftrightarrow} B\rangle_s \rightarrow B$

Rule? (acknowledge)
$\Downarrow$

$Production$

Produce rule $B \longrightarrow A: \{N_a, B\}_A$

Final Goal:
The initial goal can be realized by the following protocol:
$$\begin{array}{l} A \longrightarrow B: \{N_a, A\}_B \\ B \longrightarrow A: \{N_b, B\}_A \\ A \longrightarrow B: \{N_b, A\}_B \\ B \longrightarrow A: \{N_a, B\}_A \end{array}$$
Rule? (extract protocol)

Solution

$$\boxed{\begin{array}{l} A \longrightarrow B: \{N_a, A\}_B \\ B \longrightarrow A: \{N_b, N_a, B\}_A \\ A \longrightarrow B: \{N_b, A\}_B \end{array}}$$

The rule *split* is a propositional rule. The rule *assert* allows assertion of a formula that is, either an initial condition of the protocol or an already-realized formula. Finally, the *extraction* rule collects the generated protocol actions, orders them, and produces the protocol.

Implementation

We have implemented a simple procedure for the automatic generation of security protocols as an interactive theorem prover. From the conjunction of the protocol goals, the user can invoke a set of commands allowing the derivation of a protocol that satisfies the set of goals. The prototype is implemented in the ocaml language (Ocaml2000). A new implementation built on the PVS theorem prover is under way. The use of PVS allows us to use its typechecker and its powerful specification language. PVS (Owre *et al.* 1999) also implements different useful instantiation and propositional simplification rules. A set of new rules is added to the prover and allows us to derive from the proof the set of protocol rules.

Discussion

We have sketched a method for automatically generating security protocols from their logical specification. We use the well-known BAN logic (Burrows, Abadi, & Needham 1990) to describe protocol goals, and we extend the logic with protocol derivation rules that allow the derivation of messages from logical statements. The correctness of the derivation rules is justified using a trace model that allows us to characterize the state where a certain formula is valid as a state in a particular sequence of events that corresponds to the messages observed so far. Our method is correct. That is, all rules are correct and any protocol generated will satisfy its security properties. However, we do not address the issue of completeness. It will be interesting to take advantage of the decidability of modal logics such as the BAN logic to provide complete methods for the generation of protocols.

Acknowledgement

I'm grateful for useful criticisms and suggestions made by the anonymous referees and by my colleagues Bruno Dutertre, Bob Riemenschneider, Victoria Stavridou, and Tomás Uribe.

References

Abadi, M., and Gordon, A. D. 1999. A calculus for cryptographic protocols: The spi calculus. *Information and Computation* 148(1):1–70.

Abadi, and Tuttle. 1991a. A semantics for a logic authentication (extended abstract). In *PODC: 10th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*.

Abadi, M., and Tuttle, M. R. 1991b. A semantics for a logic of authentication. In Logrippo, L., ed., *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, 201–216. Montéal, Québec, Canada: ACM Press.

Burrows, M.; Abadi, M.; and Needham, R. 1990. A logic of authentication. *ACM Transactions on Computer Systems* 8(1):18–36.

Buttyán, L.; Staamann, S.; and Wilhelm, U. 1998. A simple logic for authentication protocol design. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*. Rockport, MA, USA: IEEE Computer Society Press.

Clark, J., and Jacob, J. 2000. Searching for a solution: Engineering tradeoffs and the evolution of provably secure protocols. In *RSP: 21th IEEE Computer Society Symposium on Research in Security and Privacy*.

Denker, G., and Saïdi, H. 2001. Middleware of security services for embedded systems. Technical report, System Design Laboratory, SRI International, Menlo Park, CA.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning about Knowledge*. Cambridge, Massachusetts: The MIT Press.

Gong, L.; Needham, R.; and Yahalom, R. 1990. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 234–248. Oakland, CA: IEEE Computer Society, Technical Committee on Security and Privacy.

Halpern, J. Y., and Zuck, L. D. 1992. A little knowledge goes a long way: Knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM* 39(3):449–478.

Menezes, A. J.; van Oorschot, P. C.; and Vanstone, S. A. 1996. *Handbook of applied cryptography*. CRC Press series on discrete mathematics and its applications. CRC Press. ISBN 0-8493-8523-7.

Millen, and Ruess. 2000. Protocol-independent secrecy. In *RSP: 21th IEEE Computer Society Symposium on Research in Security and Privacy*.

Monniaux, D. 1999. Decision procedures for the analysis of cryptographic protocols by logics of belief. In *12th Computer Security Foundations Workshop*. Mordano, Italy: IEEE Computer Society.

The Objective Caml language. `http://caml.inria.fr/`.

Owre, S.; Shankar, N.; Rushby, J. M.; and Stringer-Calvert, D. W. J. 1999. *PVS System Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA.

Paulson, L. C. 1998. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security* 6:85–128.

Perrig, A., and Song, D. 2000a. Looking for diamonds in the desert – extending automatic protocol generation to three-party authentication and key agreement protocols. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press.

Perrig, A., and Song, D. 2000b. A first step towards the automatic generation of security protocols. In *Symposium on Network and Distributed Systems Security (NDSS '00)*, 73–83. San Diego, CA: Internet Society.

Song, D. 1999. Athena: A new efficient automatic checker for security protocol analysis. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press.

Syverson, P., and van Oorschot, P. C. 1994. On unifying some cryptographic protocol logics. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 14–28. Oakland, CA: IEEE Computer Society, Technical Committee on Security and Privacy.

Thayer, J.; Herzog, J.; and Guttman, J. 1998. Honest ideals on strand spaces. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW '98)*, 66–78. Washington - Brussels - Tokyo: IEEE.

Zhou, J. 1996. *Non-repudiation*. Ph.D. Dissertation, University of London.