Formal Methods for Assuring Security of Protocols

SUSAN OLDER AND SHIU-KAI CHIN

Center for Systems Assurance, Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244, USA
Email: sueo@ecs.syr.edu, skchin@syr.edu

Establishing the security of a system is an intricate problem with subtle nuances: it requires a careful examination of the underlying assumptions, abstractions, and possible actions. Consequently, assuring that a system behaves securely is virtually impossible without the use of rigorous analytical techniques. In this article, we focus on a single cryptographic protocol (Needham–Schroeder) and show how several different formal methods can be used to identify its various vulnerabilities. These vulnerabilities include susceptibility to freshness attacks and impersonations.

Received 31 May 2000; accepted 11 September 2000

1. INTRODUCTION

Security is an intricate property that is achieved by a combination of sufficiently strong cryptographic algorithms and protocols, correct implementation of hardware and software, and appropriate assumptions about trusted authorities. Assuring that all of these factors are present and correctly integrated to form a secure system is difficult—if not impossible—without the use of rigorous and formal analytical techniques.

The purpose of this article is to give an overview of some formal methods whose goal is to lend assurance that a system using cryptographic protocols will behave securely. These methods rely on mathematical logic and are accessible to engineers. Because security must be addressed from a variety of viewpoints, it is important to use a variety of methods, each suited for addressing a particular viewpoint.

In this article, we focus on a single key-distribution protocol—the Needham–Schroeder public-key protocol [1]—and highlight how three different formal methods can be used to analyse it. Each method is suited for reasoning about a different aspect of this protocol. Two of the methods—the BAN belief logic [2] and a logic for authentication [3]—are special-purpose logics designed specifically for reasoning about certain security-related properties. The third method—the process algebra CSP [4]—is a general-purpose language for describing and reasoning about protocols with emergent behaviour. Taken together, these systems highlight the very subtle nature of security properties and the need for a variety of views into even a single protocol. They also illustrate how formal methods can identify weaknesses and hidden assumptions underlying security protocols.

The rest of this article is organized as follows. Section 2 introduces the basic notions and properties of security, and Section 3 introduces the Needham–Schroeder public-key protocol. The next three sections provide three distinct analyses of this protocol: Section 4 describes how the BAN belief logic can be used to highlight implicit assumptions about the *freshness* of information; Section 5 sketches how the process algebra CSP was used to identify a previously unknown attack on the protocol; and Section 6 gives an overview of a logic of authentication that establishes which cryptographic keys are associated with people and processes. We conclude in Section 7.

2. SECURITY PRIMER

Principals are people, keys, processes or machines that send and receive information and that access information resources (databases, processors, printers, etc.). Security properties describe the ability of principals to access information or resources. Key security properties include:

- *privacy or confidentiality*: knowing with accuracy which principals can read data;
- *integrity:* detecting corruption of information;
- *authentication:* knowing the identity of a principal or the source of information;
- *access control:* restricting the use of a resource to privileged principals;
- *non-repudiation:* preventing principals from subsequently denying their actions;
- *availability of service:* guaranteeing authorized principals always have access to services.

The *Handbook of Applied Cryptography* [5] describes each of these properties in more detail.

2.1. Public-key infrastructure

In public-key cryptography, encryption and decryption are provided through pairs of related keys: each principal has both a *public key* (which is made known to others) and a

private key (which should be known only to the principal). These keys act as mutual inverses: anything encoded with one of these keys may be decoded with the other. Thus public-key cryptography supports both privacy and digital signatures.

If principal A wishes to send a secret message M to principal B, she encrypts M with B's public key: intuitively, only B has knowledge of his private key and hence only B can decrypt the message. In contrast, if A wishes to *sign* a message M, she encrypts it with her own *private* key: intuitively, anyone with access to her public key can verify her signature, but no one should be able to forge her signature without knowledge of her private key.

A public-key infrastructure (PKI) supports the distribution, management and use of public keys and certificates to provide authentication, privacy and other security properties. PKIs are based on certification authorities that vouch for the integrity of cryptographic information and they form the basis of current Internet security.

2.2. Basic security foundations

A system's overall security depends on several items, including:

- the cryptographic strength of the system (e.g., the computational infeasibility of decrypting messages without the proper keys);
- the protocols built on top of the cryptographic algorithms (e.g., the secure sockets layer (SSL) protocol used by web browsers);
- the correct association of specific cryptographic keys with specific principals.

Cryptographic strength is assessed over time by a combination of complexity analysis and resistance to cryptanalysis [6]. We do not address the use of formal methods to analyse cryptographic strength in this article.

Assessing security protocols often involves an analysis of ways to defeat a protocol by using bits of previously successful protocol runs (known as replay attacks) or by some form of impersonation (e.g., 'man in the middle' attacks). These analyses are particularly important, because these protocols are run repeatedly over time and typically over public networks: many security protocols depend critically on the freshness of secrets. We illustrate this type of analysis in Sections 4 and 5.

Associating cryptographic keys with principals is typically done using certificates that are digitally signed by recognized certification authorities, as in the X.509 public-key certificate standard [7]. Determining the public key of a principal is done using a chain of certificates that enable one principal to move from one certificate authority to another in a secure (i.e. digitally signed and checked) way to fetch and check the certificate of another principal. The network of certification authorities is referred to as a *trust model* or *trust topology*. An example analysis of a trust topology appears in Section 6.

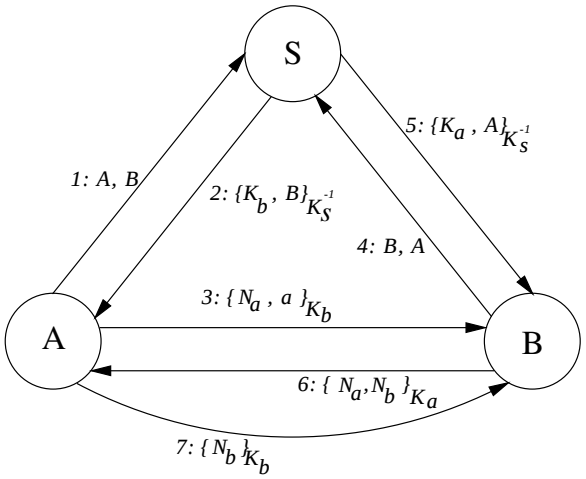

FIGURE 1. Needham–Schroeder protocol.

1. $A \rightarrow S$: A, B
2. $S \rightarrow A$: $\{K_b, B\}_{K_s^{-1}}$
3. $A \rightarrow B$: $\{N_a, A\}_{K_b}$
4. $B \rightarrow S$: B, A
5. $S \rightarrow B$: $\{K_a, A\}_{K_s^{-1}}$
6. $B \rightarrow A$: $\{N_a, N_b\}_{K_a}$
7. $A \rightarrow B$: $\{N_b\}_{K_b}$

FIGURE 2. Needham–Schroeder message exchanges.

3. NEEDHAM–SCHROEDER PROTOCOL

An important class of security protocols is the class of key-distribution protocols that enable secure communication sessions where both parties are active at the same time. Examples of such sessions include secure remote logins, secure file transfers and secure electronic-commerce transactions.

The Needham–Schroeder protocol [1] is designed to allow two principals to mutually authenticate themselves through a series of message exchanges, as a prelude for some secure session. A diagram of the message exchanges appears in Figure 1, and Figure 2 details the message exchanges in linear form.

In these descriptions, the principals are A, B and S, where S is a certification authority (CA) recognized by principals A and B. Their public keys are (respectively) K_a, K_b, and K_s; the private key of S corresponding to the public key K_s is K_s^{-1}. N_a and N_b are *nonce* identifiers: a nonce is a fresh value created for the current run of a protocol, which has not been seen in previous runs. We use common notation for describing security protocols: $A \rightarrow B$ denotes principal A sending a message to principal B; comma (',') denotes conjunction or concatenation (e.g., 'X, Y' denotes a message containing both X and Y); and $\{X\}_K$ denotes the message X encrypted using key K.

Principal A—wishing to communicate with principal B—must get B's public key from S and then convey a nonce N_a to B (messages 1–3). B then needs to get A's public

key from S and convey a nonce N_b to A in such a way that convinces A that she is interacting with B (messages 4–6). Finally, A must convince B that he is interacting with A and that A has received the nonce N_b (message 7).

Messages 1 and 2—as well as messages 4 and 5—correspond to certificate requests from A and B to an authority S for the other's public key. As messages 1 and 4 are transmitted in the clear, they have no cryptographic significance. In message 3, the nonce N_a is conveyed to B. Nevertheless, B cannot know for sure who sent N_a: he must assume that the identifier A is correct. In message 6, B sends to A both nonce N_a and N_b to identify himself as B to A, as well as to convey the nonce N_b to A. The basic notion operating here is that A and B can identify shared secrets (the nonces N_a and N_b) to each other to convince each of the other's identity.

This protocol has several weaknesses, which are addressed in the subsequent sections. The principals in the protocol implicitly assume that messages 2 and 5 are fresh and are not replays of messages containing compromised keys; this situation is analysed in Section 4 on *belief logics*. The protocol is also vulnerable to a 'man in the middle' attack, which is demonstrated in Section 5.

4. BELIEF LOGICS

One approach to reasoning about key-exchange protocols is to analyse what principals *believe* about important components and properties of protocols. These components include secret and public keys, encrypted messages, messages combined with secrets, and nonces (objects created for a specific run of a protocol). Important properties include freshness (the property of never having been used in a prior run of a protocol), jurisdiction or authority over keys, and binding of secret and public keys to principals.

The Burroughs, Abadi and Needham (BAN) logic [2] is representative of several belief logics [8, 9] that support reasoning about these properties. It focuses on proving goals such as '*A believes K_b is B's public key*'. One of the central concerns that BAN logic addresses is the possibility of *replay attacks*, where messages sent during previously successful protocol runs are re-used or replayed to trick principals into using compromised keys or to dupe principals into thinking an intruder is a legitimate participant.

The notion of time in the BAN logic is very simple. Time is divided into two categories: the *present* (i.e. the *current* run of the protocol) and the *past* (i.e. any protocol run preceding the current run).

4.1. Syntax and semantics

We present only the subset of BAN logic's syntax and semantics necessary to describe part of the Needham–Schroeder protocol. A complete description of BAN logic appears in [2]. The logic includes the following types of statements.

- $P \models X$, read as '*Principal P believes X*'. P behaves as if X is true.
- $P \lhd X$, read as '*P sees X*'. A principal has sent P a message containing X.
- $P \mid\sim X$, read as '*P once said X*'. P at some time (either in the present or the past) believed X and sent it as part of a message.
- $P \Rightarrow X$, read as '*P has jurisdiction over X*'. Principal P has authority over X and is trusted on this matter.
- $\sharp(X)$, read as '*X is fresh*'. X has not appeared in a past run of the protocol.
- $\overset{K}{\mapsto} P$, read as '*P has public key K*'. The corresponding *private key* is denoted by K^{-1} and assumed to be known only by P.

While there are over 19 inference rules that define the semantics of the BAN logic, the following three inference rules capture the essence of the logic for public-key applications.

(1) *Message-Meaning Rule for Public Keys.* If P believes that Q's public key is K, and if P sees a message X encrypted with Q's private key K^{-1}, then P believes Q once said X:

$$\frac{P \models \overset{K}{\mapsto} Q \qquad P \lhd \{X\}_{K^{-1}}}{P \models (Q \mid\sim X)}.$$

This rule uses the belief in the association of a public key with a principal to establish the source of a message.

(2) *Nonce-Verification Rule.* If P believes that X is fresh (i.e. is new to the current protocol run) and that Q once said X, then P believes that Q believes X in the current run of the protocol:

$$\frac{P \models \sharp(X) \qquad P \models (Q \mid\sim X)}{P \models (Q \models X)}.$$

This rule uses the belief about the freshness of a message (usually based on nonces), coupled with knowing the message source, to establish that a principal uttered a message in the current protocol run.

(3) *Jurisdiction Rule.* If P believes that Q has authority over X and that Q believes X, then P will believe in X:

$$\frac{P \models (Q \Rightarrow X) \qquad P \models (Q \models X)}{P \models X}.$$

This rule applies to certification authorities. A principal will adopt an authority's belief if that principal recognizes the certification authority. For example, if $P \models (Q \models \overset{K_b}{\mapsto} B)$ and $P \models (Q \Rightarrow (\overset{K_b}{\mapsto} B))$, then $P \models \overset{K_b}{\mapsto} B$. That is, P believes K_b is B's public key when P recognizes Q as a key certification authority.

4.2. Analysis of the protocol

When the Needham–Schroeder protocol is analysed using the BAN logic, two possible weaknesses come to light. In steps 2 and 5 of the protocol, certification authority S

sends to A and B (respectively) the messages $\{K_b, B\}_{K_s^{-1}}$ and $\{K_a, A\}_{K_s^{-1}}$; that is, A and B both receive the public key of the principal with whom they wish to communicate.

After A receives $\{K_b, B\}_{K_s^{-1}}$ in step 2, the desired result on A's behalf is $A \models (\overset{K_b}{\mapsto} B)$: A believes that K_b is B's public key. From the Jurisdiction Rule, this requires two other beliefs:

$$A \models (S \Rightarrow (\overset{K_b}{\mapsto} B)) \tag{1}$$

$$A \models (S \models (\overset{K_b}{\mapsto} B)). \tag{2}$$

That is, A must believe that S has authority over K_b and that S believes K_b is B's public key.

The first belief corresponds to A believing that S is a certification authority having the proper authority to certify B's cryptographic information. This is a reasonable assumption. Establishing the second belief requires the application of both the Message-Meaning and Nonce-Verification Rules.

It is reasonable to assume that A knows S's public key (i.e. $A \models (\overset{K_s}{\mapsto} S)$). From the Message-Meaning Rule and the receipt of certificate $\{K_b, B\}_{K_s^{-1}}$ from S, we can establish that

$$A \models (S \mid\sim (\overset{K_b}{\mapsto} B)).$$

That is, A believes that S once said that K_b was B's public key.

We can then use the Nonce-Verification Rule to establish that $A \models (S \models (\overset{K_b}{\mapsto} B))$, *provided* that $A \models \sharp(\overset{K_b}{\mapsto} B)$. This proviso highlights a potential weakness, as there is nothing in message 2 that corresponds to a nonce: A must assume that anything with the public key from the authority S is fresh. Consequently, the protocol is potentially vulnerable to a replay attack based on reusing old keys.

A similar analysis holds true for A's public key in step 5.

5. PROCESS ALGEBRAS

Process algebras such as CSP [4, 10] and CCS [11] provide a way to describe system behaviour in terms of the *events* (i.e. abstract actions deemed 'observable') that can occur. The collection of events that a process can engage in is known as its *alphabet*, typically denoted by Σ.

The *trace model* of CSP formally describes a process's behaviour in terms of the set of *traces*—that is, sequences of events—that it can perform. These trace sets provide a basis for comparing, equating and refining processes. Two processes are *trace-equivalent* when they have precisely the same sets of traces. A process Q *refines* P in the trace model (written $P \sqsubseteq Q$) if every trace of Q is also a trace of P. Intuitively, if P corresponds to a specification of permissible behaviour and Q refines P, then Q is guaranteed to exhibit only permissible behaviours.

Using the trace model and the FDR2 model checker [12], Lowe uncovered a previously unknown flaw in the Needham–Schroeder protocol [13, 14]. We provide an informal description of his approach in this section, first introducing the relevant CSP notation; we refer the reader to Lowe's original papers for more comprehensive accounts.

$$P \overset{\epsilon}{\Longrightarrow} P \qquad \frac{P \overset{\alpha}{\Longrightarrow} P_1 \quad P_1 \overset{\beta}{\Longrightarrow} P_2}{P \overset{\alpha\cdot\beta}{\Longrightarrow} P_2}$$

$$(e \to P) \overset{e}{\Longrightarrow} P$$

$$P_1 \sqcap P_2 \overset{\epsilon}{\Longrightarrow} P_1 \qquad P_1 \sqcap P_2 \overset{\epsilon}{\Longrightarrow} P_2$$

$$\frac{P_1 \overset{e}{\Longrightarrow} P_1' \qquad e \notin X}{P_1 \ [|X|] \ P_2 \overset{e}{\Longrightarrow} P_1' \ [|X|] \ P_2}$$

$$\frac{P_2 \overset{e}{\Longrightarrow} P_2' \qquad e \notin X}{P_1 \ [|X|] \ P_2 \overset{e}{\Longrightarrow} P_1 \ [|X|] \ P_2'}$$

$$\frac{P_1 \overset{e}{\Longrightarrow} P_1' \quad P_2 \overset{e}{\Longrightarrow} P_2' \quad e \in X}{P_1 \ [|X|] \ P_2 \overset{e}{\Longrightarrow} P_1' \ [|X|] \ P_2'}$$

$$\frac{P \overset{e}{\Longrightarrow} P' \qquad e \notin X}{P\backslash X \overset{e}{\Longrightarrow} P'} \qquad \frac{P \overset{e}{\Longrightarrow} P' \qquad e \in X}{P\backslash X \overset{\epsilon}{\Longrightarrow} P'}$$

$$\frac{P \overset{e_i}{\Longrightarrow} P'}{P[e_1 \leftarrow e_1', \ldots, e_n \leftarrow e_n'] \overset{e_i'}{\Longrightarrow} P'[e_1 \leftarrow e_1', \ldots, e_n \leftarrow e_n']}$$

$$\frac{P \overset{e}{\Longrightarrow} P' \qquad e \notin \{e_1, \ldots, e_n\}}{P[e_1 \leftarrow e_1', \ldots, e_n \leftarrow e_n'] \overset{e}{\Longrightarrow} P'[e_1 \leftarrow e_1', \ldots, e_n \leftarrow e_n']}$$

FIGURE 3. Transition relations for CSP.

5.1. CSP syntax and semantics

Informally, CSP processes (ranged over by P) can have the following forms.[1]

- STOP is the process that does nothing.
- $e \to P$ performs event e and then behaves like P.
- $P_1 \sqcap P_2$ non-deterministically chooses to behave like P_1 or to behave like P_2.
- $P_1 \ [|X|] \ P_2$ is a parallel composition of P_1 and P_2, where the set of events X constrains certain actions. Any event in the set X can occur *only* if P_1 and P_2 both perform it simultaneously; in contrast, events in $\Sigma - X$ occur only as independent actions of P_1 or P_2.
- $P\backslash X$ behaves like P, but the events in X are *hidden* (i.e. considered unobservable).
- $P[e_1 \leftarrow e_1', \ldots, e_n \leftarrow e_n']$ behaves like P, except that each event e_i is relabelled to e_i'.
  As a special case, in $P[e \leftarrow e_1, e \leftarrow e_2]$, the event e is replaced by a non-deterministic choice between the events e_1 and e_2.

Formally, we introduce a family of relations $\overset{\alpha}{\Longrightarrow}$ that describe processes' trace behaviour, writing $P \overset{\alpha}{\Longrightarrow} P'$ to indicate that process P can perform the trace α and become

[1] We introduce only those operators necessary for the rest of this section.

the process P'. This family of relations can be defined as the smallest relations satisfying the axioms and inference rules in Figure 3. The first two rules are primarily bookkeeping rules: the first expresses that every process, by doing nothing (ϵ denotes the empty trace), remains unchanged; the second reflects that computations' intermediate states can be abstracted away ($\cdot$ denotes sequence concatenation). The remaining rules are syntax-directed and formalize the informal explanations of the different process terms given previously. In particular, there is no explicit rule for STOP, because STOP never does anything.

By way of illustration, consider the following simple processes:

$$Q \equiv a \rightarrow b \rightarrow c \rightarrow \text{STOP}$$
$$R \equiv c \rightarrow c \rightarrow \text{STOP}$$

The following series of examples illustrates these definitions.

(1) Q has precisely four traces: $\langle\rangle$, $\langle a\rangle$, $\langle a, b\rangle$, $\langle a, b, c\rangle$.
Likewise, R has precisely three: $\langle\rangle$, $\langle c\rangle$, $\langle c, c\rangle$.

(2) The traces of $Q \sqcap R$ are simply those of Q and R: $\langle\rangle$, $\langle a\rangle$, $\langle a, b\rangle$, $\langle a, b, c\rangle$, $\langle c\rangle$, $\langle c, c\rangle$.

(3) The parallel composition $Q\ [|\{b, d\}|]\ R$ has the following traces: $\langle\rangle$, $\langle a\rangle$, $\langle c\rangle$, $\langle a, c\rangle$, $\langle c, a\rangle$, $\langle c, c\rangle$, $\langle a, c, c\rangle$, $\langle c, a, c\rangle$, $\langle c, c, a\rangle$.
Note that Q, denied cooperation from R, is unable to perform its b event.

(4) In contrast, $Q\ [|\{c, d\}|]\ R$ has these four traces: $\langle\rangle$, $\langle a\rangle$, $\langle a, b\rangle$, $\langle a, b, c\rangle$.
R must delay c events until Q is ready to cooperate; furthermore, R can only perform *one* c event, because Q provides only one opportunity to do so.

(5) The traces of $(Q \sqcap R)\backslash\{b, d\}$ are as follows: $\langle\rangle$, $\langle a\rangle$, $\langle a, c\rangle$, $\langle c\rangle$, $\langle c, c\rangle$.

(6) Finally, $(Q \sqcap R)[b \leftarrow a, c \leftarrow e, c \leftarrow d]$ has precisely these traces: $\langle\rangle$, $\langle a\rangle$, $\langle a, a\rangle$, $\langle a, a, d\rangle$, $\langle a, a, e\rangle$, $\langle d\rangle$, $\langle e\rangle$, $\langle d, d\rangle$, $\langle d, e\rangle$, $\langle e, d\rangle$, $\langle e, e\rangle$.
Note that every occurrence of b in a trace of $Q \sqcap R$ is replaced by a; every occurrence of c is replaced *either* by d or by e.

In practice, it is useful to have events with multiple fields: for example, a description of a bank account might use events such as *deposit.50*, *deposit.100* and *withdraw.100*. Furthermore, a bank account should be prepared to accept deposits of any amount: the notation *deposit*?x represents a choice among all *deposit* events and the variable x becomes bound to the actual value of the second field. Thus, for example, the process $in?w \rightarrow out.w \rightarrow$ STOP has among its traces $\langle in.13, out.13\rangle$ and $\langle in.45, out.45\rangle$.

5.2. Modelling the protocol's messages

For simplicity of presentation, we consider a simplified version of the protocol in which A and B already know each other's public keys. Under these circumstances, A and B can forgo the communications with the key server S, resulting in a protocol in which only three messages need to be exchanged:

1. $A \rightarrow B$: $\{N_a, A\}_{K_b}$
2. $B \rightarrow A$: $\{N_a, N_b\}_{K_a}$
3. $A \rightarrow B$: $\{N_b\}_{K_b}$

This simplification does not affect the final result of the analysis: the flaw uncovered also exists in the seven-message version of the protocol.

In the analysis that follows, three forms of compound events will represent these messages:

1. $msg1.A.B.Encrypt.K_b.N_a.A$
2. $msg2.B.A.Encrypt.K_a.N_a.N_b$
3. $msg3.A.B.Encrypt.K_b.N_b$

In each case, the event captures the message's role in the protocol, its sender and receiver, the key used to encrypt, and the contents of the encrypted message.

These events are further extended with prefixes (*comm*, *intercept*, *fake*) to represent messages that are communicated safely between A and B, intercepted by an intruder, or forged, respectively.

```
INITIATOR(a, n_a) =
Irunning.a?b →comm.msg1.a.b.Encrypt.K_b.n_a.A
             →comm.msg2.b.a.Encrypt.K_a?n?nb
             →if not (n == n_a)
                 then STOP
                 else (comm.msg3.a.b.Encrypt.K_b.nb
                     →Icommit.a.b
                     →session.a.b →STOP)
```

FIGURE 4. The process INITIATOR(a, n_a).

5.3. Modelling the agents

A CSP description for a generic initiator of this protocol appears in Figure 4. Intuitively, the process INITIATOR(a, n_a) represents an initiator a who originally possesses the single nonce n_a and has public key K_a. The event *Irunning.a*?*b* represents a's intent to initiate a run of the protocol with some agent b; the portion '?*b*' of this event indicates that b can be instantiated to any valid principal of the system. (There is a similar event *Rrunning.b*?a' that represents a recipient b's belief that it is engaged in a run of the protocol with a principal claiming to be a'.) The initiator's transmission of the first message in the protocol is represented by the event *comm.msg1.a.b.Encrypt.*K_b*.*n_a*.a*, where K_b is the public key associated with b; the initiator then waits for the response message from b, as represented by the event[2] *comm.msg2.b.a.Encrypt.*K_a?*n*?*nb*.

The initiator terminates the protocol if the nonce n received is not the nonce that it originally transmitted; this termination is represented by the deadlock process STOP.

[2]The '?*n*?*nb*' portion of this event indicates that the parameters n and nb will be instantiated based on the particular nonces that appear in b's response message.

$$\begin{aligned}\text{INIT} = \text{INITIATOR}(A, N_A)\ [&\mathit{comm.msg1} \leftarrow \mathit{comm.msg1},\\ &\mathit{comm.msg1} \leftarrow \mathit{intercept.msg1},\\ &\mathit{comm.msg2} \leftarrow \mathit{comm.msg2},\\ &\mathit{comm.msg2} \leftarrow \mathit{fake.msg2},\\ &\mathit{comm.msg3} \leftarrow \mathit{comm.msg3},\\ &\mathit{comm.msg3} \leftarrow \mathit{intercept.msg3}]\end{aligned}$$

FIGURE 5. The revised initiator INIT.

If the nonce *does* match, then the initiator transmits the final message to b (per the event $\mathit{comm.msg3.a.b.Encrypt.K_b.nb}$) and commits to the protocol (represented by the event *Icommit.a.b*). The final event *session.a.b* abstracts the actual session between a and b.

As given, the process $\text{INITIATOR}(a, n_a)$ does not incorporate the possibility of intercepted or faked messages. We can include these possibilities via a simple renaming, as in Figure 5, where we also fix a particular initiator A with nonce N_A. This renaming captures the notion that the Initiator's communications can be intercepted and that the messages it receives could be forged by an intruder, unbeknown to the Initiator (i.e. the Initiator behaves as if every communication is legitimate).

A responder process can be written in a similar fashion, after which the pair of agents are placed in parallel:

$$\text{AGENTS} = \text{INIT}\ [|\mathit{comm}, \mathit{session}|]\ \text{RESP}$$

Here, *comm* and *session* are used to constrain the behaviour of the processes: *comm* and *session* events may occur only if both agents participate in them simultaneously.

5.4. Modelling the intruder

In modelling the intruder, it pays to be as general as possible: if we encode particular types of attacks, then at best we can argue that the protocol is invulnerable to those specific attacks. Instead, we encode only the following general (and standard) assumptions about the intruder:

(1) the intruder can potentially overhear and intercept any message in the system;
(2) the intruder can decrypt any message encoded with her own public key;
(3) the intruder can replay intercepted messages (and alter any plaintext components of them), even if she cannot decrypt the message itself;
(4) the intruder can introduce new messages into the system, using any nonces she has available to her.

In particular, we assume that the intruder *cannot* decrypt messages encrypted with keys that she does not possess. However, we make no assumptions about the intruder's status within the network: the intruder may be an insider or an outsider.

The complete CSP description is fairly straightforward but rather long. As a simplification that illustrates the approach, Figure 6 provides a CSP description for an intruder in a system where the only messages being passed around are of the second type (i.e. those that contain two nonces encrypted); the true description of the intruder process also incorporates *comm*, *fake* and *intercept* events involving messages 1 and 3. The process $\text{INTR}(Ms, Ns)$ is parameterized by two sets: Ms, which contains the undecrypted messages the intruder has intercepted so far, and Ns, the set of nonces she has collected so far. The *comm* events correspond to communications in which the intruder is a legitimate participant (i.e. the intruder is the original sender or intended recipient). The *fake* events correspond to replays that the intruder performs: she can replay any message that she has already seen,[3] and she can modify the plaintext fields (i.e. the identities of senders and receivers) at will. Finally, the *intercept* events correspond to message interceptions: if the message is encoded with her own key, the intruder can decrypt it and add the two (now decrypted) nonces to her collection of known nonces; if not, then the message becomes one that she can replay at a later date.

$$\begin{aligned}&\text{INTR}(Ms, Ns) =\\ &\quad \mathit{comm.msg2?b.a.Encrypt.k.n1.n2}\\ &\qquad \rightarrow \text{if } (k == K_i)\\ &\qquad\quad \text{then INTR}(Ms, Ns \cup \{n1, n2\})\\ &\qquad\quad \text{else INTR}(Ms \cup \{\mathit{Encrypt.k.n1.n2}\}, Ns)\\ &\quad \sqcap\ \mathit{fake.msg2?a?b?m{:}Ms} \rightarrow \text{INTR}(Ms, Ns)\\ &\quad \sqcap\ \mathit{fake.msg2?a?b.Encrypt?k?n} : Ns?n' {:} Ns \rightarrow \text{INTR}(Ms, Ns)\\ &\quad \sqcap\ \mathit{intercept.msg2?b.a.Encrypt.k.n1.n2}\\ &\qquad \rightarrow \text{if } (k == K_i)\\ &\qquad\quad \text{then INTR}(Ms, Ns \cup \{n1, n2\})\\ &\qquad\quad \text{else INTR}(Ms \cup \{\mathit{Encrypt.k.n1.n2}\}, Ns)\end{aligned}$$

FIGURE 6. The intruder process $\text{INTR}(Ms, Ns)$.

Finally, the entire system is defined by placing the agents and an intruder with nonce N_I in parallel:

$$\begin{aligned}&\mathit{SYSTEM} =\\ &\qquad \text{AGENTS}\ [|\mathit{comm, fake, intercept}|]\ \text{INTR}(\emptyset, \{N_I\})\end{aligned}$$

5.5. Specifying and assessing authentication

Authentication requires that whenever a principal A thinks she has established a session with B, B has indeed been running the protocol with A. Because we wish to verify two-way authentication, there are two properties to specify: (1) the initiator A commits to the session only if the responder B thinks he has participated in the protocol with A; and (2) the responder B commits to the session only if the initiator A thinks she has participated in the protocol with B.

These two properties can be expressed in CSP as the processes *RA* (*receiver authentication*) and *IA* (*initiator authentication*):

$$\begin{aligned}RA &= \mathit{Rrunning.A.B} \rightarrow \mathit{Icommit.A.B} \rightarrow RA\\ IA &= \mathit{Irunning.A.B} \rightarrow \mathit{Rcommit.A.B} \rightarrow IA\end{aligned}$$

[3] The '$?m{:}Ms$' portion of the event indicates that m may be instantiated only by a member of Ms.

To verify that the authentication properties hold, it suffices to perform the following two refinement checks:

$$RA \sqsubseteq SYSTEM \backslash (\Sigma - \{Rrunning.A.B, Icommit.A.B\})$$
$$IA \sqsubseteq SYSTEM \backslash (\Sigma - \{Irunning.A.B, Rcommit.A.B\})$$

That is, if every trace of *SYSTEM* (with all events other than *Rrunning.A.B* and *Icommit.A.B* hidden) is also a trace of *RA*, then the property *RA* is guaranteed to hold (and similarly for the property *IA*).

It turns out that the first refinement check succeeds, but the second check fails: there are traces of *SYSTEM* in which the event *Rcommit.A.B* occurs without a preceding *Irunning.A.B* event. For example, *SYSTEM* can perform the following sequence of events:

Irunning.A.I	(1)
intercept.msg1.A.I.Encrypt.K_I*.*N_A*.A*	(2)
fake.msg1.A.B.Encrypt.K_B*.*N_A*.A*	(3)
Rrunning.A.B	(4)
intercept.msg2.B.A.Encrypt.K_A*.*N_A*.*N_B	(5)
fake.msg2.I.A.Encrypt.K_A*.*N_A*.*N_B	(6)
intercept.msg3.A.I.Encrypt.K_I*.*N_B	(7)
fake.msg3.A.B.Encrypt.K_B*.*N_B	(8)
Rcommit.A.B	(9)

The existence of this trace highlights a potential 'man in the middle' attack, whereby the intruder participates as a legitimate recipient in one run of the protocol with A (the events in lines (1)–(2) and (6)–(7)) while impersonating A in a second run of the protocol with B (lines (3)–(5) and (8)–(9)). Ultimately, B commits to a session with A, despite the fact that A never even attempted to interact with B.

6. ASSOCIATING KEYS AND PRINCIPALS

Up until now, we have taken for granted the association between principals and keys. Correctly making this association is crucial, particularly when using a PKI. Associating keys with principals typically depends on two components.

(1) *Certification authorities:* principals who are recognized as having the authority to vouch for the correctness of the associations between keys and principals.
(2) *Certificates:* data structures that associate keys with principals. They are digitally signed by certification authorities to preserve the integrity of the cryptographic information.

Certificates are much like driver's licenses and certification authorities are like the network of authorities who issue those licenses. Chief among standards for public-key authentication services is the X.509 standard [7].

Figure 7 contains an example of a very simple multiple certification-authority (MCA) network, where principal *CA1* is the certification authority for principal A, *CA2* is the certification authority for principal B, and *CA1* has a certificate for *CA2*. In this example, using the notation of X.509, the following certificates exist.

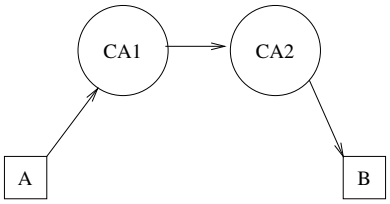

FIGURE 7. Multiple certification authorities.

- *CA1*$\langle\langle$*CA2*$\rangle\rangle$: a certificate digitally signed with *CA1*'s private key certifying that key $CA2_p$ is *CA2*'s public key. The integrity of this certificate is checked using *CA1*'s public key $CA1_p$.
- *CA2*$\langle\langle B\rangle\rangle$: a certificate digitally signed with *CA2*'s private key certifying that key B_p is B's public key. The integrity of this certificate is checked using *CA2*'s public key $CA2_p$.

Informally, principal A can get principal B's public key using the above certificates, provided that A knows *CA1*'s public key (i.e. $CA1_p$). First, A uses $CA1_p$ and the certificate *CA1*$\langle\langle$*CA2*$\rangle\rangle$ to get an integrity-checked copy of *CA2*'s public key ($CA2_p$). In the notation of X.509, this step is represented by the equation

$$CA2_p = CA1_p \bullet CA1\langle\langle CA2\rangle\rangle,$$

where $\bullet$ denotes the operator that extracts a key from a certificate and checks its integrity using the supplied public key. Using $CA2_p$, A then extracts from the certificate $CA2\langle\langle B\rangle\rangle$ an integrity-checked copy of B_p:

$$B_p = CA2_p \bullet CA2\langle\langle B\rangle\rangle.$$

6.1. Reasoning about certification

Lampson *et al.* [3] created a logic to reason about authentication in distributed systems. One of the main purposes of the logic is to answer questions such as 'Who is speaking?' and 'Whom does this key speak for or represent?'. While space considerations do not allow a complete description of the logic here, we give a flavour of the logic and its use by examining the MCA example with it.

There are two central notions necessary for our analysis.

(1) *Principals making statements.* This notion is denoted by P *says* s, where P is a principal and s is some (logical) statement. Implicit in *says* statements is the notion that the statement can be traced back to some digitally signed statement. Principals can be people, machines, operating systems or cryptographic keys.
(2) *Principals speaking for principals.* The notion of principal P speaking for principal Q is denoted by $P \Rightarrow Q$. A common idiom is $P_p \Rightarrow P$, which indicates

that P's public key P_p speaks for P: statements that are digitally signed using P_p will be inferred to be statements made by P.

The logic is defined by approximately 19 axioms, including the following *handoff axiom*:

$$(P \textit{ says } (Q \Rightarrow P)) \supset (Q \Rightarrow P).$$

This handoff axiom says that whenever a principal P states that another principal Q speaks on P's behalf, then Q does speak on P's behalf. Related to this axiom is the property

$$(P \Rightarrow Q) \supset ((P \textit{ says } s) \supset (Q \textit{ says } s)).$$

This property—derivable from the logic's axioms—states that whenever a principal P speaks for Q and makes a statement s, it is safe to behave as if Q made the statement s.

6.2. Analysis of the MCA example

Returning to the example of Figure 7, we show how this logic helps us to reason about the certificates and the implicit trust assumptions that underlie A's trust in B_p as B's public key. That is, we can isolate sufficient assumptions under which A can conclude that $B_p \Rightarrow B$.

It turns out that the following five assumptions are sufficient for concluding $B_p \Rightarrow B$.

(1) $CA1_p \Rightarrow CA1$. A knows the public key of its certification authority (i.e. from A's perspective, the key $CA1_p$ speaks for $CA1$).
(2) $CA1_p$ *says* $(CA2_p \Rightarrow CA2)$. $CA1$'s public key $CA1_p$ is used to verify the validity of the certificate $CA1\langle\langle CA2\rangle\rangle$.
(3) $CA2_p$ *says* $(B_p \Rightarrow B)$. Similarly, the key $CA2_p$ is used to verify the certificate $CA2\langle\langle B\rangle\rangle$.
(4) $CA1 \Rightarrow CA2$. A trusts its certificate authority to speak for other certificate authorities ($CA2$ in this case).
(5) $CA2 \Rightarrow B$. A knows that B's certificate authority is $CA2$.

The proof of $B_p \Rightarrow B$ from these assumptions is straightforward. From the derived property $(P \Rightarrow Q) \supset ((P \textit{ says } s) \supset (Q \textit{ says } s))$ and assumptions (1) and (2), we can deduce that $CA1$ *says* $(CA2_p \Rightarrow CA2)$. Likewise, we can then use assumption (4) to deduce that $CA2$ *says* $(CA2_p \Rightarrow CA2)$.

From the handoff axiom, we get that $CA2_p \Rightarrow CA2$. At this point, the derived property and the certificate for B in assumption (3) allow us to conclude that $CA2$ *says* $(B_p \Rightarrow B)$. Another application of the derived property, this time using assumption (5), lets us deduce that B *says* $(B_p \Rightarrow B)$. Finally, the handoff axiom lets us conclude that $B_p \Rightarrow B$, which was our original goal.

The value of this analysis is twofold: (1) it makes explicit the trust assumptions being made, and (2) it assures a consistent treatment of certificates and assumptions about certification authorities.

7. CONCLUSIONS

In this article, we have not attempted to give an exhaustive description of the applications of formal methods to the problem of assuring security: we have examined only a single session-based protocol, designed to provide mutual authentication using a PKI. In particular, we have ignored shared-key cryptographic systems as well as *store-and-forward* security (e.g., secure electronic mail).

Instead, we have focused on a single protocol and sketched how multiple logical systems and formal models can provide insight into security issues from a variety of viewpoints. In our experience, the choice of a particular formalism is driven in part by the properties one wishes to prove and also by the tools available. Both the BAN logic of Section 4 and the authentication of logic of Section 6 are special-purpose modal logics, specifically designed to support reasoning about freshness and trust. They focus attention on what principals must be prepared to accept and to believe in order to trust in the correctness of a protocol. In contrast, the process algebra CSP described in Section 5 is a general-purpose language for describing and reasoning about the behaviour of concurrent systems. For this reason, it is well suited for reasoning about the high-level interactions and events that may occur during a run of a protocol.

These analyses demonstrate the subtlety of security properties and the importance of having rigorous methods for assessing the security of a system. Security properties are affected by timing and timeliness (e.g., present versus past runs of a protocol), trust (or lack thereof) in various principals and authorities, and cryptographic properties. Furthermore, there are nuances that arise between different levels of abstraction. The value of these formal methods is that they help in the detection of weaknesses and possible attacks, as well as making explicit any necessary assumptions that have been made.

Finally, the methods described in this article are all accessible to engineers with an understanding of predicate logic. We have taught each of these formalisms to both computer-science and computer-engineering students. The specialized logics can be embedded into theorem provers: [15] describes the embedding of a BAN-like logic into the HOL theorem prover [16]. Likewise, there are tools such as Lowe's Casper [17] that automatically translate abstract descriptions of security protocols into process-algebraic descriptions that can be analysed with model checkers.

ACKNOWLEDGEMENTS

This work was supported in part by the New York State Center for Advanced Technology in Computer Applications and Software Engineering (CASE).

A special acknowledgement

This paper is written in honour of Graham Birtwistle's 60th birthday. For decades, he held a series of formal-methods workshops in the stimulating environment of

the Canadian Rocky Mountains. These intellectually challenging workshops included logicians, mathematicians, computer scientists and engineers. The authoring team of Older (logician) and Chin (engineer) is a reflection of the kind of research community inspired by Graham. We are all better off because of his efforts.

REFERENCES

[1] Needham, R. and Schroeder, M. (1978) Using encryption for authentication in large networks of computers. *Commun. ACM*, **21**, 993–999.

[2] Burroughs, M., Abadi, M. and Needham, R. (1989) *A Logic of Authentication*. Report 39, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, February.

[3] Lampson, B., Abadi, M., Burroughs, M. and Wobber, E. (1992) Authentication in distributed systems: theory and practice. *ACM Trans. Comput. Syst.*, **10**, 265–310.

[4] Hoare, C. A. R. (1985) *Communicating Sequential Processes* (*Series in Computer Science*). Prentice-Hall, London.

[5] Menzies, A. J., von Oorschot, P. C. and Vanstone S. A. (1996) *Handbook of Applied Cryptography*. CRC Press, New York.

[6] Landau, S. (2000) Standing the test of time: the data encryption standard. *Notices AMS*, **47** 341–349.

[7] ITU-T Recommendation X.509 (1993) Also ISO/EC 9594-8 (1995) *Information Technology—Open Systems Interconnection—The Directory: Authentication Framework*.

[8] Gong, L., Needham, R. and Yahalom, R. (1990) Reasoning about belief in cryptographic protocols. In *Proc. 1990 IEEE Comput. Soc. Symp. on Research in Security and Privacy*, Oakland, CA, May, pp. 234–248. IEEE Society Press, Washington, DC.

[9] van Oorschot, P. (1993) Extending cryptographic logics of belief to key agreement protocols. In *Proc. First ACM Conf. on Computers and Communications Security*, Fairfax, VA, November, pp. 232–243. ACM Press, New York.

[10] Roscoe, A. W. (1998) *The Theory and Practice of Concurrency* (*Series in Computer Science*). Prentice-Hall, London.

[11] Milner, R. (1980) *A Calculus of Communicating Systems*, Vol. 92. Springer, Berlin.

[12] Formal Systems (Europe) Ltd. (1997) *Failures-Divergence Refinement: FDR2 User Manual*, Oxford.

[13] Lowe, G. (1995) An attack on the Needham–Schroeder public-key authentication protocol. *Inform. Process. Lett.*, **56**, 131–133.

[14] Lowe, G. (1996) Breaking and fixing the Needham–Schroeder public-key protocol using fdr. *Software Concepts Tools*, **17**, 93–102.

[15] Schubert, T. and Mocas, S. (1995) A mechanized logic for secure key escrow protocol verification. In *Proc. Eighth Int. Workshop on Higher Order Logic Theorem Proving and Its Applications*.

[16] Gordon, M. J. C. and Melham, T. F. (1993) *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, Cambridge.

[17] Lowe, G. (1998) Casper: a compiler for the analysis of security protocols. *J. Comput. Security*, **6**, 53–84.