# I/O Automaton Models and Proofs for Shared-Key Communication Systems

Nancy Lynch
MIT Laboratory for Computer Science
Cambridge, MA 02139, USA
lynch@lcs.mit.edu

## Abstract

*The combination of two security protocols, a simple shared-key communication protocol and the Diffie-Hellman key distribution protocol, is modeled formally and proved correct. The modeling is based on the I/O automaton model for distributed algorithms, and the proofs are based on invariant assertions, simulation relations, and compositional reasoning. Arguments about the cryptosystems are handled separately from arguments about the protocols.*

## 1. Introduction

Security protocols must satisfy important correctness requirements, which means that it is important to be able to think about them clearly and precisely. But they can also be large and complicated, which makes such reasoning difficult. One needs ways of decomposing the task into clearly separable pieces. This includes separating different types of concerns, for example, distributed algorithms issues, cryptosystem computability issues, probabilistic issues, and issues of accurate modeling of reality. It also includes decomposing the protocols using the normal techniques for decomposing distributed algorithms, based on levels of abstraction and parallel composition of interacting components.

This paper describes an experiment in modeling and analyzing security protocols, using *I/O automata* [14, 12] and the usual techniques that go along with them—a combination of invariant assertions, simulation relations, and compositional reasoning using traces. The aim of the experiment is to explore how these methods can help in decomposing the task of reasoning about security protocols. This model and these methods have been used successfully for decomposing the reasoning about many standard distributed algorithms (see, e.g., [12, 16, 13]), and about several distributed system designs (see, e.g., [6, 7, 9, 10]), so it is worth discovering what they can do for security protocols.

The experiment involves combining simple shared-key communication and key distribution protocols to implement private communication. In the case we describe in detail here, simple Diffie-Hellman key distribution [4] is used, the protocols tolerate only passive eavesdroppers, and only safety properties are considered. In another case in progress, discussed briefly here, the more complex Diffie-Oorschot-Weiner key distribution protocol [5], which tolerates adversaries that can intrude more actively, is studied. Later work will include liveness guarantees, formulated in terms of timing properties.

Our main guideline in studying these protocols is to try to *decompose the reasoning* as much as possible, identifying sub-problems that can be treated separately. (Although the examples in this paper are simple enough to be understood informally, we believe that understanding how best to decompose them is a good first step toward understanding how to decompose more complex examples.) The handling of each piece should be appropriately abstract. For example, in discussing protocol issues, cryptosystem computability issues should be summarized by assumptions saying that certain values are not "easily computable" from others; number-theoretic arguments about *why* these values are not (likely to be) easily computable should be treated at a lower level, as mechanisms to achieve the more abstract non-computability guarantees. Probabilistic issues should be treated separately, as far as possible. After dividing up the problems in this way, we expect that the main benefit of the I/O automaton-based methods will be in clarifying the distributed algorithm issues. Cryptosystem issues, for example, may be better treated by other means, for example, the inductive techniques of Paulson [15]. However, a general framework should provide a rigorous way of combining the different types of issues.

In treating the distributed algorithms themselves, we similarly try to decompose them as much as possible. The most obvious form of decomposition involves treating the two sub-protocols separately, then trying to paste them together using general theorems about automaton composition. Another form involves giving very high level automaton specifications for services, giving separate descriptions of implementing algorithms, and showing, by means of simula-

tion relations, that the algorithms implement the services. Still another form involves first studying a protocol using a natural, simple cryptosystem, and later trying to show that its correctness properties extend to modified versions that use more elaborate cryptosystems. And still another form of decomposition involves combining adversaries that interact with separate protocols into a single "colluding" unit.

Because I/O automata are composed by means of shared actions, and because we are considering only safety properties in this paper, it is natural to describe external behavior of automata in terms of sets of *traces* (i.e., sequences of external actions). The simple trace semantics yields simple and powerful projection and pasting theorems (see, e.g., [12], p. 211), for the behavior of compositions of automata. However, in order to enable compositional reasoning about particular kinds of properties, the traces must contain all the information relevant for those properties. For example, in treating fault-tolerance properties such as *wait-free termination* and *$f$-failure termination* compositionally, in terms of traces, it is convenient to allow the traces to contain special $fail$ input actions that signal the occurrence of failure events (see, e.g., [12, 13]). Sometimes it is convenient to consider different strengths of failure actions (e.g., the $good$, $bad$, and $ugly$ failure actions in [7]). Also, in order to treat timing properties compositionally, it is useful to introduce timing information into the traces.

In the case of security protocols, important properties involve *lack of knowledge*. To treat this compositionally, one should include something about knowledge in the traces. Our approach here is to give explicit $learn$ input actions and $reveal$ output actions by which a component can learn new information and reveal its knowledge, and to constrain the component's behavior in terms of these actions.

Specifically, the paper contains the following. Section 2 contains math preliminaries. Section 3 presents a model for cryptosystems, which describe the data types encountered in the protocols, including (cleartext and ciphertext) messages, keys, and lower-level data from which keys are constructed. This data model also describes the functions that manipulate data, and the reachability (computability) relationships that say which values can be computed easily from which others. This model is similar to others in the literature. Section 4 then describes some "standard" types of automata that model certain components appearing in many systems—service environments, insecure channels, and eavesdroppers.

Section 5 gives I/O automaton specifications for the two main security services considered in this paper—private communication and key distribution. The specification for private communication is abstract: it talks only about communication and revealed information, and not about keys. Section 6 models and analyzes the implementation of private communication using an abstract key distribution service, and Section 7 treats the Diffie-Hellman implementation of key distribution. These protocols use particular cryptosystems, and the protocol proofs assume the limitations on easy computability expressed by those cryptosystems. The proofs are based on invariant assertions and on simulation relations relating the protocols to the specifications for the services they are intended to implement.

Section 8 shows what is involved in moving from a description of each of the two individual protocols in terms of its own natural cryptosystem to a description in terms of a common, richer cryptosystem. For example, the shared key protocol is initially analyzed in terms of abstract, unstructured keys taken from a simple "shared-key cryptosystem". However, when one combines this protocol with Diffie-Hellman, it is necessary to consider a version that uses structured keys, taken from a richer "structured-key cryptosystem".

Section 9 puts the pieces together, to get an implementation of private communication that uses shared-key communication with Diffie-Hellman key distribution. Most of this is accomplished automatically from the general projection and pasting theorems for I/O automata; special arguments must be made for combining the insecure channels used in the two protocols, and for combining the two adversaries into one. Section 10 gives a final discussion. Because of limited space, most of the proofs have been omitted from this version; the rest will appear in a technical report [].

**Related work:** Of the formal work on modeling and analyzing cryptographic protocols, the efforts that seem closest in spirit to ours are those of Abadi and of Paulson. Abadi has developed a framework for formal study of composable security protocols [2, 1], and Paulson has developed inductive reasoning methods, which appear valuable both for proving assertions and for determining cryptosystem reachability relationships [15]. Others have stated and proved invariant assertions for security protocols, though we do not know of other work on simulation relations for such protocols. We also do not know about other work using traces with explicit $learn$ and $reveal$ actions as an approach to compositional reasoning about security protocols. Wing and Cheiner are currently modeling security protocols using the approach of this paper, including verifying assertions using the PVS theorem prover.

**Acknowledgments:** I thank Ron Rivest for getting me started on this project and for some very helpful discussions about this project. Martin Abadi, Oleg Cheiner, Butler Lampson, Victor Luchangco, Anna Lysyanskaya, Dalia Malkhi, Mike Reiter, and Jeannette Wing provided useful comments and encouragement.

## 2. Mathematical Preliminaries

$\lambda$ denotes the empty string. We use I/O automata as defined in [12]. Since we do not deal with liveness in this paper, the tasks are irrelevant. If $A$ and $B$ are I/O automata with the same external signature, then we say that $A$ *implements* $B$ provided that every trace of $A$ is also a trace of $B$. Invariants and simulation relations are defined, for example, in [12].

## 3. Data Model

This section gives a basic model for the data types used in the protocols.

### 3.1. Cryptosystems

A *cryptosystem signature* $\mathcal{S}$ consists of:

- $TN_{\mathcal{S}}$, a set of *type names*.
- $FN_{\mathcal{S}}$, a set of *function names*.
- $domain_{\mathcal{S}}$, a mapping from $FN_{\mathcal{S}}$ to $(TN_{\mathcal{S}})^*$.
- $range_{\mathcal{S}}$, a mapping from $FN_{\mathcal{S}}$ to $TN_{\mathcal{S}}$.
- $EN_{\mathcal{S}} \subseteq FN_{\mathcal{S}}$, a set of *easy* function names.

A *constant name* is a function name $f$ such that $domain_{\mathcal{S}}(f) = \lambda$. Let $CN_{\mathcal{S}} \subseteq FN_{\mathcal{S}}$ denote the set of constant names of $\mathcal{C}$. We omit the subscript $\mathcal{S}$ where no confusion seems likely. A *cryptosystem* $\mathcal{C}$ consists of:

- A cryptosystem signature $sig_{\mathcal{C}}$. We write $TN_{\mathcal{C}}$ as shorthand for $TN_{sig_{\mathcal{C}}}$, etc.
- $set_{\mathcal{C}}$, a mapping from $TN_{\mathcal{C}}$ to disjoint sets.
- $fun_{\mathcal{C}}$, a mapping from $FN_{\mathcal{C}}$ to functions; We require that if $domain_{\mathcal{C}}(f) = (t_1, \ldots, t_k)$ and $range_{\mathcal{C}}(f) = t$ then $fun_{\mathcal{C}}(f) : set_{\mathcal{C}}(t_1) \times \cdots \times set_{\mathcal{C}}(t_k) \rightarrow set_{\mathcal{C}}(t)$.

We write $set_{\mathcal{C}}$ for $\bigcup_{t \in TN_{\mathcal{C}}} set_{\mathcal{C}}(t)$. We omit the subscript $\mathcal{C}$ where no confusion seems likely. If $X \cup \{y\} \subseteq set_{\mathcal{C}}$, we say that $y$ is *easily reachable* from $X$ in $\mathcal{C}$ provided that $y$ is obtainable starting from elements of $X$, by applying only functions denoted by function names in $EN_{\mathcal{C}}$.

### 3.2. Term Cryptosystems

If $\mathcal{S}$ is a cryptosystem signature, then the *terms* of $\mathcal{S}$, and their *types*, are defined recursively, as follows:

1. If $c \in CN_{\mathcal{S}}$ and $range_{\mathcal{S}}(c) = t$, then $c$ is a term and $type_{\mathcal{S}}(c) = t$.
2. If $f \in FN_{\mathcal{S}}$, $domain_{\mathcal{S}}(f) = t_1, t_2, \ldots, t_k$, where $k \geq 1$, $range_{\mathcal{S}}(f) = t$, and $e_1, \ldots, e_k$ are terms of types $t_1, \ldots, t_k$, respectively, then the expression $e = f(e_1, \ldots, e_k)$ is a term, and $type_{\mathcal{S}}(e) = t$.

Let $Terms_{\mathcal{S}}(t)$ denote the set of terms of $\mathcal{S}$ of type $t$. Let $Terms_{\mathcal{S}}$ denote the set of all terms of $\mathcal{S}$.

Some of the cryptosystems we consider are best understood as term algebras derived from cryptosystem signatures. In these cases, the values of the various types are, formally, equivalence classes of terms: An equivalence relation $R$ on $Terms_{\mathcal{S}}$ is said to be a *congruence* provided that the following hold.

1. If $eRe'$ then $type_{\mathcal{S}}(e) = type_{\mathcal{S}}(e')$.
2. Suppose that $f \in FN_{\mathcal{S}}$, $domain_{\mathcal{S}}(f) = t_1, t_2, \ldots, t_k$, where $k \geq 1$, $range_{\mathcal{S}}(f) = t$, $e_1, \ldots, e_k$ are terms of types $t_1, \ldots, t_k$, respectively, $e'_1, \ldots, e'_k$ are terms of types $t_1, \ldots, t_k$, respectively, and for all $i$, $1 \leq i \leq k$, $e_i R e'_i$. Then $f(e_1, \ldots, e_k) R f(e_1, \ldots, e_k)$.

Let $\mathcal{S}$ be a cryptosystem signature and $R$ a congruence on $Terms_{\mathcal{S}}$. Then the *term cryptosystem* $\mathcal{C}$ for $\mathcal{S}$ and $R$ is the unique cryptosystem satisfying:

- $sig_{\mathcal{C}} = \mathcal{S}$.
- If $t \in TN_{\mathcal{C}}$, then $set_{\mathcal{C}}(t)$ is the set of all $R$-equivalence classes of terms of type $t$ in $Terms_{\mathcal{C}}$.
- If $f \in FN_{\mathcal{C}}$, $domain_{\mathcal{C}}(f) = (t_1, \ldots, t_k)$ and $range_{\mathcal{C}}(f) = t$ then $fun_{\mathcal{C}}(f)$ is the function from $set_{\mathcal{C}}(t_1) \times \cdots \times set_{\mathcal{C}}(t_k)$ to $set_{\mathcal{C}}(t)$ defined as follows. Suppose that $e_i \in set_{\mathcal{C}}(t_i)$ for all $i$, $1 \leq i \leq k$. Then $fun_{\mathcal{C}}(f)([e_1]_R, \ldots, [e_k]_R)$ is defined to be $[f(e_1, \ldots, e_k)]_R$. (Since $R$ is a congruence, this is well-defined.)

We use the notation $R_{\mathcal{C}}$ for the congruence relation $R$ of $\mathcal{C}$. If $e \in Terms_{\mathcal{C}}$, then we write $[e]_{\mathcal{C}}$ for the equivalence class of $e$ with respect to $R_{\mathcal{C}}$. Also, if $E \subseteq Terms_{\mathcal{C}}$ then we write $[E]_{\mathcal{C}}$ for the set of equivalence classes $[e]_{\mathcal{C}}$ for $e \in E$.

### 3.3. Cryptosystem Examples

In this subsection we give the cryptosystems used later in the paper. The first kind of cryptosystem, a shared-key cryptosystem, is used in shared key communication. The second kind, a base-exponent cryptosystem, is used in the Diffie-Hellman protocol. The third kind, a structured-key cryptosystem, is essentially a combination of the two others. It is used when the two protocols are combined.

#### 3.3.1 Shared-key cryptosystems

A *shared-key* cryptosystem $\mathcal{C}$ is a term cryptosystem. The signature $\mathcal{S} = sig_{\mathcal{C}}$ is defined as follows. $TN_{\mathcal{S}}$ consists of two type names: "$M$" for messages and "$K$" for keys. $FN_{\mathcal{S}}$ consists of:

- $enc$, with $domain(enc) = (\text{“}M\text{”}, \text{“}K\text{”})$ and $range(enc) = \text{“}M\text{”}$.
- $dec$, with $domain(dec) = (\text{“}M\text{”}, \text{“}K\text{”})$ and $range(dec) = \text{“}M\text{”}$.
- $MConst_{\mathcal{S}}$, a set of message constant names, with $range(m) = \text{“}M\text{”}$ for all $m \in MConst_{\mathcal{S}}$.
- $KConst_{\mathcal{S}}$, a set of key constant names, with $range(k) = \text{“}K\text{”}$ for all $k \in KConst_{\mathcal{S}}$.

$EN_{\mathcal{S}} = \{enc, dec\}$. The relation $R$ is defined by means of all equations of the form:

- $dec(enc(m,k),k) = m$, where $m, k \in Terms_{\mathcal{S}}$, $type(m) = \text{“}M\text{”}$, $type(k) = \text{“}K\text{”}$.

Specifically, we want the smallest congruence relation on $Terms_{\mathcal{S}}$ that equates all terms that are related by the given equations.

#### 3.3.2 Base-exponent cryptosystems

A *base-exponent* cryptosystem $\mathcal{C}$ is a term cryptosystem in which, letting $\mathcal{S} = sig_{\mathcal{C}}$: $TN_{\mathcal{S}}$ consists of two type names, "$B$" for bases and "$X$" for exponents. $FN_{\mathcal{S}}$ consists of:

- $exp$, with $domain(exp) = (\text{“}B\text{”}, \text{“}X\text{”})$ and $range(exp) = \text{“}B\text{”}$.
- $BConst_{\mathcal{S}}$, a set of base constant names, with $range(b) = \text{“}B\text{”}$ for all $b \in BConst_{\mathcal{S}}$.
- $XConst1_{\mathcal{S}}$ and $XConst2_{\mathcal{S}}$, two disjoint sets of exponent constant names, with $domain(x) = \lambda$ and $range(x) = \text{“}X\text{”}$ for all $x \in XConst1_{\mathcal{S}} \cup XConst2_{\mathcal{S}}$.

$EN_{\mathcal{S}} = \{exp\} \cup BConst_{\mathcal{S}}$. The relation $R$ is defined by means of all equations of the form:

- $exp(exp(b,x),y) = exp(exp(b,y),x)$, where $b, x, y \in Terms_{\mathcal{S}}$, $type(b) = \text{“}B\text{”}$, $type(x) = type(y) = \text{“}X\text{”}$.

Define $B2_{\mathcal{S}}$ to be the set of all terms of the form $exp(exp(b,x),y)$, where $b \in BConst_{\mathcal{S}}$, $x \in XConst1_{\mathcal{S}}$ and $y \in XConst2_{\mathcal{S}}$. An *augmented base-exponent* cryptosystem is a base-exponent cryptosystem together with a distinguished element $b0_{\mathcal{S}}$ of $BConst_{\mathcal{S}}$.

#### 3.3.3 Structured-key cryptosystems

A structured-key cryptosystem is a combination of a shared-key cryptosystem and a base-exponent cryptosystem, where certain terms of the base-exponent cryptosystem are identified with the keys. A *structured-key* cryptosystem $\mathcal{C}$ is a term cryptosystem in which, letting $\mathcal{S} = sig_{\mathcal{C}}$: $TN_{\mathcal{S}}$ consists of the type names "$M$", "$B$", and "$X$". $FN_{\mathcal{S}}$ consists of:

- $enc$, with $domain(enc) = (\text{“}M\text{”}, \text{“}B\text{”})$ and $range(enc) = \text{“}M\text{”}$.
- $dec$, with $domain(dec) = (\text{“}M\text{”}, \text{“}B\text{”})$ and $range(dec) = \text{“}M\text{”}$.
- $exp$, with $domain(exp) = (\text{“}B\text{”}, \text{“}X\text{”})$ and $range(exp) = \text{“}B\text{”}$.
- $MConst_{\mathcal{S}}$, a set of message constant names, with $range(m) = \text{“}M\text{”}$ for all $m \in MConst_{\mathcal{S}}$.
- $BConst_{\mathcal{S}}$, a set of base constant names, with $range(b) = \text{“}B\text{”}$ for all $b \in BConst$.
- $XConst1_{\mathcal{S}}$ and $XConst2_{\mathcal{S}}$, two disjoint sets of exponent constant names, with $range(x) = \text{“}X\text{”}$ for all $x \in XConst1_{\mathcal{S}} \cup XConst2_{\mathcal{S}}$.

$EN_{\mathcal{S}} = \{enc, dec, exp\} \cup BConst_{\mathcal{S}}$. The relation $R$ is defined by means of all equations of the form:

- $dec(enc(m,b),b) = m$, where $m, b \in Terms_{\mathcal{S}}$, $type(m) = \text{“}M\text{”}$, $type(b) = \text{“}B\text{”}$.
- $exp(exp(b,x),y) = exp(exp(b,y),x)$, where $b, x, y \in Terms_{\mathcal{S}}$, $type(b) = \text{“}B\text{”}$, $type(x) = type(y) = \text{“}X\text{”}$.

Once again, we write $B2_{\mathcal{S}}$ for the set of terms of the form $exp(exp(b,x),y)$, where $b \in BConst_{\mathcal{S}}$, $x \in XConst1_{\mathcal{S}}$, and $y \in XConst2_{\mathcal{S}}$. An *augmented structured-key* cryptosystem is a structured-key cryptosystem together with a distinguished element $b0_{\mathcal{S}}$ of $BConst_{\mathcal{S}}$.

## 4. Some Generally-Useful Automata

In this section, we give automaton models for some system components that will appear in many settings: environments for security services, insecure channels, and eavesdroppers. They are presented in a parameterized fashion so that they can be used in different contexts. We model these components as automata (rather than, e.g., by trace properties) for uniformity with the way we will model algorithms and system specs, and because this makes it possible to reason about them assertionally.

### 4.1. Environment Automata

Here we assume that $U$ is a universal set of data values, $A$ is an arbitrary finite set of adversary ports, that is, locations where information can be communicated to the adversary, and $N \subseteq U$. The environment automaton $Env(U, A, N)$ models any entities other than the channels from which an eavesdropper may learn information. It says that the environment is capable of communicating elements of $U$ at any adversary port $a \in A$, but in fact does not communicate any elements of $N$.

$Env(U, A, N)$ :

**Signature:**

| Input: | Output: |
|---|---|
| None | $learn(u)_a$, $u \in U, a \in A$ |

**States:**

No variables

**Transitions:**

$learn(u)_a$
- Precondition:
  - $u \notin N$
- Effect:
  - $none$

### 4.2. Insecure Channel Automata

Here we assume that $U$ is a universal set of data values, $P$ is an arbitrary finite set of client ports, and $A$ is an arbitrary finite set of adversary ports. The insecure channel admits $send$ and $receive$ actions for all elements of $U$ and also has $eavesdrop$ output actions, by which information in transit passes to an outsider. The insecure channel allows any message in transit to be communicated to an outsider.

$IC(U, P, A)$:

**Signature:**

Input:
- $IC\text{-}send(u)_{p,q}$, $u \in U, p, q \in P, p \neq q$

Output:
- $IC\text{-}receive(u)_{p,q}$, $u \in U, p, q \in P, p \neq q$
- $eavesdrop(u)_{p,q,a}$, $u \in U, p, q \in P, p \neq q, a \in A$

**States:**

for every $p, q \in P, p \neq q$:
- $buffer(p, q)$, a multiset of $U$, initially empty

**Transitions:**

$IC\text{-}send(u)_{p,q}$
- Effect:
  - add $u$ to $buffer(p, q)$

$IC\text{-}receive(u)_{p,q}$
- Precondition:
  - $u \in buffer(p, q)$
- Effect:
  - remove one copy of $u$ from $buffer(p, q)$

$eavesdrop(u)_{p,q,a}$
- Precondition:
  - $u \in buffer(p, q)$
- Effect:
  - $none$

### 4.3. Eavesdropper Automata

Here we assume that $\mathcal{C}$ is a cryptosystem, $P$ is an arbitrary finite set of client ports, and $A$ is an arbitrary finite set of adversary ports. We define a model for an eavesdropper, as a nondeterministic automaton $Eve(\mathcal{C}, P, A)$. $Eve$ simply remembers everything it learns and hears, and can reveal anything it has, at any time. It does this by maintaining a variable $has$, initially $\emptyset$. The value of $has$ may change only in restricted ways: Namely, when $eavesdrop(u)_{p,q,a}$ or $learn(u)_a$ occurs, $u$ gets added to $has$. When an internal $compute$ action occurs, the value resulting from applying an easy function (one in $EN_{\mathcal{C}}$) to values in $has$ may be added to $has$. We restrict the $reveal(u)$ output so that $u \in has$, that is, $Eve$ can only report a value that it has.

$Eve(\mathcal{C}, P, A)$:

**Signature:**

Input:
- $eavesdrop(u)_{p,q,a}$, $u \in set_{\mathcal{C}}, p, q \in P, p \neq q, a \in A$
- $learn(u)_a$, $u \in set_{\mathcal{C}}, a \in A$

Output:
- $reveal(u)_a$, $u \in set_{\mathcal{C}}, a \in A$

Internal:
- $compute(u, f)_a$, $f \in EN_{\mathcal{C}}, a \in A$

**States:**

$has \subseteq set_{\mathcal{C}}$, initially $\emptyset$

**Transitions:**

$eavesdrop(u)_{p,q,a}$
- Effect:
  - $has := has \cup \{u\}$

$learn(u)_a$
- Effect:
  - $has := has \cup \{u\}$

$reveal(u)_a$
- Precondition:
  - $u \in has$
- Effect:
  - $none$

$compute(u, f)_a$
- Precondition:
  - $\{u_1, \ldots, u_k\} \subseteq s.has$
  - $u = f(u_1, \ldots, u_k)$
- Effect:
  - $has := has \cup \{u\}$

## 5. The Services

In this section, we describe the two services that are implemented by the protocols in this paper. They are described as automata, which is convenient for assertional reasoning. The use of input and output actions provides convenient ways of composing these automata with others, and of describing what is preserved by implementation relationships. For simplicity, we write these specifications to describe only safety properties, although the same methods can be used to handle liveness properties, formulated as time bounds (see, e.g., [11, 12]).

### 5.1. Private Communication

This section contains a specification of the problem of achieving private communication among the members of a finite collection $P$ of clients. The specification expresses three properties: (1) only messages that are sent are delivered, (2) messages are delivered at most once each, and (3) none of the messages is revealed by an "adversary". We describe the problem using a high-level I/O automaton specification $PC(U, P, M, A)$, where $U$ is a universal set of data values, $P$ is an arbitrary finite set of client ports, $M \subseteq U$ is a set of messages, and $A$ is an arbitrary finite set of adversary ports. This specification does not mention distribution or keys; these aspects will appear in implementations of this specification, but not in the specification itself. The specification simply describes the desired properties, as an abstract machine. As usual for automaton specifications, the properties, listed separately above, are intermingled in one description.

$PC(U, P, M, A)$:

**Signature:**

Input:
$PC\text{-}send(m)_{p,q}$, $m \in M, p, q \in P, p \neq q$

Output:
$PC\text{-}receive(u)_{p,q}$, $u \in U, p, q \in P, p \neq q$
$reveal(u)_a$, $u \in U, a \in A$

**States:**

for every pair $p, q \in P$, $p \neq q$:
$buffer(p, q)$, a multiset of $M$

**Transitions:**

$PC\text{-}send(m)_{p,q}$
Effect:
add $m$ to $buffer(p, q)$

$PC\text{-}receive(u)_{p,q}$
Precondition:
$u \in buffer(p, q)$
Effect:
remove one copy of $u$ from $buffer(p, q)$

$reveal(u)_a$
Precondition:
$u \notin M$
Effect:
$none$

The first two properties listed above, which amount to at-most-once delivery of messages that were actually sent, are expressed by the transition definitions for $PC\text{-}send$ and $PC\text{-}receive$. The third property, privacy, is expressed by the constraint for $reveal$.

### 5.2. Key Distribution

This is a drastically simplified key distribution service, which distributes a single key to several participants. We do not model requests for the keys, but assume that the service generates the key spontaneously. The simplified key distribution problem is specified by the automaton $KD(U, P, K, A)$, where $U$ is a universal set of data values, $P$ is an arbitrary finite set of client ports, $K \subseteq U$ is a set of keys, and $A$ is a finite set of adversary ports.

$KD(U, P, K, A)$:

**Signature:**

Input:
none
Output:
$grant(u)_p$, $u \in U, p \in P$
$reveal(u)_a$, $u \in U, a \in A$

Internal:
$choose\text{-}key$

**States:**

$chosen\text{-}key$, an element of $K \cup \{\perp\}$, initially $\perp$
$notified \subseteq P$, initially $\emptyset$

**Transitions:**

$choose\text{-}key$
Precondition:
$chosen\text{-}key = \perp$
Effect:
$chosen\text{-}key :=$ choose $k \in K$

$grant(u)_p$
Precondition:
$chosen\text{-}key \neq \perp$
$u = chosen\text{-}key$
$p \notin notified$
Effect:
$notified :=$ $notified \cup \{p\}$

$reveal(u)_a$
Precondition:
$u \notin K$
Effect:
$none$

## 6. Implementing Private Communication using Shared Keys

This section describes a straightforward shared-key communication protocol. The protocol simply uses a shared key, obtained from a key distribution service, to encode and decode messages. Throughout the section, we assume that $\mathcal{C}$ is a shared-key cryptosystem, $P$ is a set (of clients) with at least 2 elements, and $A$ is a nonempty finite set (of adversaries).

### 6.1. The Encoder and Decoder

We define parameterized encoder and decoder automata, parameterized by the shared-key cryptosystem $\mathcal{C}$, the set $P$ of clients, and elements $p, q \in P$, $p \neq q$. Note that, in the code for $IC\text{-}send(u)$, we are using the abbreviation $enc$ for $fun_{\mathcal{C}}(enc)$ – that is, we are suppressing mention of the particular cryptosystem $\mathcal{C}$.

$Enc(\mathcal{C}, P)_{p,q}$, **where** $p, q \in P$, $p \neq q$ :

**Signature:**

Input:
$PC\text{-}send(m)_{p,q}$, $m \in [MConst_{\mathcal{C}}]$
$grant(u)_p$, $u \in set_{\mathcal{C}}$

Output:
$IC\text{-}send(u)_{p,q}$, $u \in set_{\mathcal{C}}$

**States:**

$buffer$, a multiset of elements of $[MConst_{\mathcal{C}}]$, initially empty
$shared\text{-}key \in [KConst_{\mathcal{C}}] \cup \{\perp\}$, initially $\perp$

**Transitions:**

$PC\text{-}send(m)_{p,q}$
Effect:
add $m$ to $buffer$

$grant(u)_p$
Effect:
if $u \in [KConst_{\mathcal{C}}]$ then
$shared\text{-}key := u$

$IC\text{-}send(u)_{p,q}$
Precondition:
$m$ is in $buffer$
$shared\text{-}key \neq \perp$
$u = enc(m, shared\text{-}key)$
Effect:
remove one copy of $m$
from $buffer$

More-or-less symmetrically, we have:

$Dec(\mathcal{C}, P)_{p,q}$**, where** $p, q \in P, p \neq q$ :
**Signature:**

Input:
$IC\text{-}receive(u)_{p,q}$, $u \in set_{\mathcal{C}}$
$grant(u)_q$, $u \in set_{\mathcal{C}}$

Output:
$PC\text{-}receive(u)_{p,q}$, $u \in set_{\mathcal{C}}$

**States:**

$buffer$, a multiset of elements of $set_{\mathcal{C}}$ ("$M$"), initially empty
$shared\text{-}key \in [KConst_{\mathcal{C}}] \cup \{\perp\}$, initially $\perp$

**Transitions:**

$IC\text{-}receive(u)_{p,q}$
Effect:
if $u \in set_{\mathcal{C}}$("$M$") then
add $u$ to $buffer$

$grant(u)_q$
Effect:
if $u \in [KConst_{\mathcal{C}}]$ then
$shared\text{-}key := u$

$PC\text{-}receive(u)_{p,q}$
Precondition:
$m$ is in $buffer$
$shared\text{-}key \neq \perp$
$u = dec(m, shared\text{-}key)$
Effect:
remove one copy of $m$
from $buffer$

## 6.2. The Complete Implementation

In the rest of this section, we assume: $U = set_{\mathcal{C}}$; $M = [MConst_{\mathcal{C}}]$; $K = [KConst_{\mathcal{C}}]$; $N = M \cup K$; $U'$ is an arbitrary set with $K \subseteq U'$; $A'$ is an arbitrary set, disjoint from $A$.

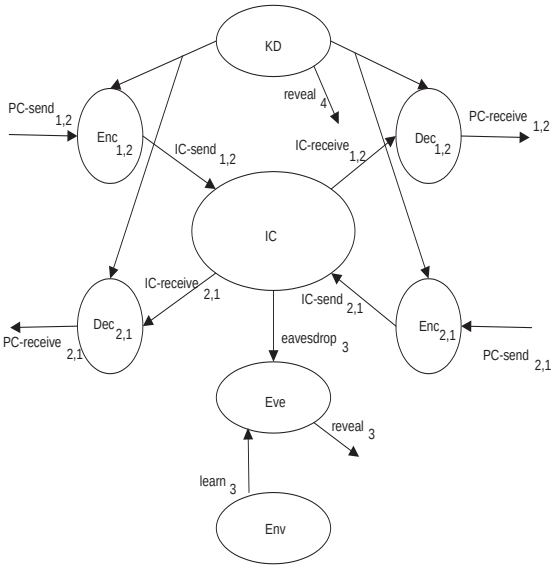


**Figure 1.** $S_1$; $P$ ={1,2}, $A$ ={3}; $A\prime$ ={4}

The implementation consists of encoder and decoder components, an insecure channel, eavesdropper and environment, plus a key distribution service. More precisely, the implementation, $S_1(\mathcal{C}, P, A, U', A')$, is obtained by composing the following automata and then hiding certain actions.

- $Enc(\mathcal{C}, P)_{p,q}$, $Dec(\mathcal{C}, P)_{p,q}$, $p, q \in P, p \neq q$.
- $IC(U, P, A)$, $Eve(\mathcal{C}, P, A)$, $Env(U, A, N)$.
- $KD(U', P, K, A')$, a key distribution service.

In this system, the eavesdropper $Eve$ does not acquire any information directly from the $KD$ component. Later, in Section 9, we combine this eavesdropper with another that arises in the key distribution service implementation.

To get $S_1(\mathcal{C}, P, A, U', A')$, we hide the following actions in the composition just defined: $eavesdrop_{p,q,a}$, $p, q \in P$, $a \in A$; $IC\text{-}send_{p,q}$, $IC\text{-}receive_{p,q}$, $p, q \in P$; $grant_p$, $p \in P$; $learn_a$, $a \in A$; $reveal_a$, $a \in A'$. We sometimes omit explicit mention of parameters of $S_1$, or of other systems and components, when we think that confusion is unlikely. Figure 1 contains an interaction diagram for $S_1$.

Our system model says that the eavesdropper learns no elements of $N = M \cup K$ from outside sources. That choice of $N$ is fine for this protocol, but we do not now have a general prescription for how to choose "good" sets $N$ for all protocols. ("Good" here means that the set should have a simple definition, should be large enough to include all values that the adversary could use to break the protocol, and should be small enough to exclude values produced by other protocols with which the given protocol is to be composed.)

### 6.3. Invariants

In system $S_1$, we use $Enc_{p,q}$, $Dec_{p,q}$, $IC$, $Eve$, and $KD$ as "handles" to help in naming state variables in the composed state. This handle naming device for state variables is taken from Vaziri's work [17]. The first invariant says that the keys granted by the key distribution service are consistent.

**Lemma 6.1** *In all reachable states of $S_1$, the following are true:*

1. *If $Enc_{p,q}.shared\text{-}key \neq \perp$ then $Enc_{p,q}.shared\text{-}key = KD.chosen\text{-}key$.*
2. *If $Dec_{p,q}.shared\text{-}key \neq \perp$ then $Dec_{p,q}.shared\text{-}key = KD.chosen\text{-}key$.*

The next invariant says that no $N$ elements appear in $Eve.has$ or in the insecure channel.

**Lemma 6.2** *In all reachable states of $S_1$, the following are true:*

1. *For all $p, q \in P$, $p \neq q$, and all $u \in N$, $u \notin IC.buffer(p,q)$.*
2. *If $u \in N$ then $u \notin Eve.has$.*

**Lemma 6.3** *In all reachable states of $S_1$, the following are true:*

1. *If $u \in N$ then $u$ is not easily reachable from $Eve.has \cup (U - N)$ in $\mathcal{C}$.*

The proofs of the first two of these invariants are straightforward inductive arguments. In some of the steps (e.g., the $IC\text{-}send$ steps in Part 1 of Lemma 6.2), facts about the cryptosystem are used (in this case, some inequivalence facts for terms). The third invariant follows from the second.

### 6.4. Implementation Proof

We show that $S_1$ implements $PC(U, P, M, A)$, using a simulation relation from $S_1$ to $PC(U, P, M, A)$. The relation $F$ is defined by saying that $(s, t) \in F$ provided that the following condition holds:
For each $p, q \in P, p \neq q, t.buffer(p,q)$ is the multiset union of three multisets, $A_1, A_2, A_3$, of $U$, where:

1. $A_1 = s.Enc_{p,q}.buffer$.
2. $A_2 = dec(s.IC.buffer(p,q), s.KD.chosen\text{-}key)$ if $s.KD.chosen\text{-}key \neq \perp$ else $\emptyset$.
3. $A_3 = dec(s.Dec_{p,q}.buffer, s.KD.chosen\text{-}key)$ if $s.KD.chosen\text{-}key \neq \perp$ else $\emptyset$.

That is, each high-level multiset of messages in transit is obtained from the messages in the buffers at the encoder and decoder, plus those in transit in the low-level insecure channels. The messages in the insecure channels and in the decoder buffer must be decoded for the correspondence.

**Theorem 6.4** *$F$ is a simulation relation.*

**Proof:** By standard assertional methods for proving simulations, see, e.g., [12], p. 225. The invariants of the preceding section are used here. □

**Theorem 6.5** *$S_1(\mathcal{C}, P, A, U', A')$ implements $PC(U, P, M, A)$.*

**Proof:** Follows from Theorem 6.4. □

## 7. Diffie-Hellman Key Distribution Protocol

This section describes the Diffie-Hellman key distribution protocol. Throughout the section, we assume $\mathcal{C}$ is an augmented base-exponent cryptosystem, $P = \{p1, p2\}$, and $A$ is a nonempty set.

### 7.1. The Endpoint Automata

We define two symmetric automata, for the two elements of $P$.

$DH(\mathcal{C}, P)_{p1}$:

**Signature:**

Input:
$IC\text{-}receive(b)_{p2,p1}$, $b \in set_{\mathcal{C}}(\text{“}B\text{”})$
Output:
$IC\text{-}send(b)_{p1,p2}$, $b \in set_{\mathcal{C}}(\text{“}B\text{”})$
$grant(b)_{p1}$, $b \in set_{\mathcal{C}}(\text{“}B\text{”})$

Internal:
$choose\text{-}exp_{p1}$

**States:**

$chosen\text{-}exp \in [XConst1_{\mathcal{C}}] \cup \{\perp\}$, initially $\perp$
$base\text{-}sent$, a Boolean, initially $false$
$rcvd\text{-}base \in set_{\mathcal{C}}(\text{“}B\text{”}) \cup \{\perp\}$, initially $\perp$
$granted$, a Boolean, initially $false$

**Derived variables:**
$chosen\text{-}base \in set_{\mathcal{C}}(\text{“}B\text{”}) \cup \{\perp\}$, given by:
if $chosen\text{-}exp \neq \perp$ then $exp([b0_{\mathcal{C}}], chosen\text{-}exp)$ else $\perp$

**Transitions:**

$choose\text{-}exp_{p1}$
Precondition:
$chosen\text{-}exp = \perp$
Effect:
$chosen\text{-}exp :=$
choose $x \in [XConst1_{\mathcal{C}}]$

$IC\text{-}send(b)_{p1,p2}$
Precondition:
$chosen\text{-}exp \neq \perp$
$b = chosen\text{-}base$
$base\text{-}sent = false$
Effect:
$base\text{-}sent := true$

$IC\text{-}receive(b)_{p2,p1}$
Effect:
$rcvd\text{-}base := b$

$grant(b)_{p1}$
Precondition:
$chosen\text{-}exp \neq \perp$
$rcvd\text{-}base \neq \perp$
$b = exp(rcvd\text{-}base, chosen\text{-}exp)$
$granted = false$
Effect:
$granted := true$

The automaton for $p2$ is the same, but interchanges uses of $p1$ and $p2$, and likewise of $XConst1$ and $XConst2$.

### 7.2. The Complete Implementation

In the rest of this section, we assume: $U = set_{\mathcal{C}}$; $K = [B2_{\mathcal{C}}]$; $X = [XConst1_{\mathcal{C}}] \cup [XConst2_{\mathcal{C}}]$; $N = K \cup X$.

The implementation consists of two endpoint automata, an insecure channel, an eavesdropper and an environment. Specifically, implementation $S_2(\mathcal{C}, P, A)$ is the composition of the following automata, with certain actions hidden:

- $DH(\mathcal{C}, P)_p$, $p \in P$, endpoint automata.
- $IC(U, P, A)$, $Eve(\mathcal{C}, P, A)$, $Env(U, A, N)$.

To get $S_2(\mathcal{C}, P, A)$, we hide: $eavesdrop_{p,q,a}$, $p, q \in P$, $p \neq q$, $a \in A$; $IC\text{-}send_{p,q}$, $IC\text{-}receive_{p,q}$, $p, q \in P$, $p \neq q$; $learn_a$, $a \in A$. Figure 2 contains an interaction diagram for $S_2$.

### 7.3. Invariants

In system $S_2$, we use $DH(p)$ for $p \in P$, $IC$, and $Eve$ as handles to help in naming state variables in the composed state. The first invariant says that messages that have been received or are in transit are correct:

**Lemma 7.1** *In all reachable states of $S_2$, the following are true:*

1. *If $DH(p).rcvd\text{-}base \neq \perp$ and $q \neq p$ then $DH(q).chosen\text{-}exp \neq \perp$, and $DH(q).rcvd\text{-}base = DH(p).chosen\text{-}base$.*
2. *If $u \in IC.buffer(p, q)$, then $DH(p).chosen\text{-}exp \neq \perp$, and $u = DH(p).chosen\text{-}base$.*

**Lemma 7.2** *In all reachable states of $S_2$, the following are true:*

1. *For all $p, q \in P$, $p \neq q$, and all $u \in N$, $u \notin IC.buffer(p, q)$.*

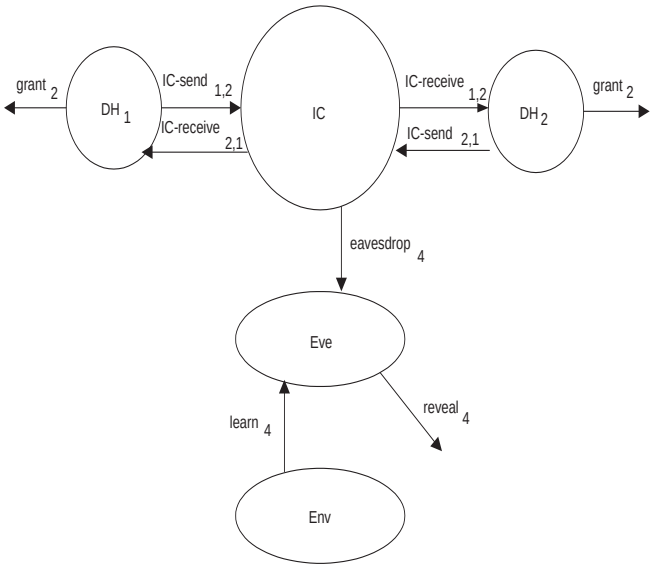


**Figure 2.** $S_2$; $P$ ={1,2}; $A$ ={4}

2. *If $u \in N$ then $u \notin Eve.has$.*

**Lemma 7.3** *In all reachable states of $S_2$, the following are true:*

1. *If $u \in N$ then $u$ is not easily reachable from $Eve.has \cup (U - N)$ in $\mathcal{C}$.*

### 7.4. Implementation Proof

We show that $S_2$ implements $KD(U, P, K, A)$ using a simulation relation. The relation $F$ is defined by saying that $(s, t) \in F$ provided that:

1. If $s.DH(p).chosen\text{-}exp \neq \perp$ for all $p \in P$, then $t.chosen\text{-}key = exp(s.DH(p1).chosen\text{-}base, s.DH(p2).chosen\text{-}exp)$, and otherwise $t.chosen\text{-}key = \perp$.
2. $t.notified = \{p \in P : s.DH(p).granted\}$.

**Theorem 7.4** *$F$ is a simulation relation.*

**Proof:** By induction.
*Base:* Easy.
*Inductive step:* Consider $(s, \pi, s')$ and $t$ and consider cases. The most interesting cases are:

1. $\pi = choose\text{-}exp_p$.

   If $s.DH(q).chosen\text{-}exp = \perp$, where $q \neq p$ then this maps to the trivial one-state execution fragment $t$. The correspondence is trivially preserved (part 1 is vacuous). Otherwise, this corresponds to $choose\text{-}key$,

with a chosen value of $exp(s'.DH(p1).chosen\text{-}base, s'.DH(p2).chosen\text{-}exp)$. Enabling is straightforward, as is the preservation of the simulation.

2. $\pi = grant(b)_p$

   This corresponds to $grant(b)_p$ in the specification. The interesting fact to show here is the enabling, specifically, that the value $b = exp(s.DH(p).rcvd\text{-}base, s.DH(p).chosen\text{-}exp)$ is equal to $t.chosen\text{-}key$. But Lemma 7.1 implies that $b = exp(s.DH(q).chosen\text{-}base, s.DH(p).chosen\text{-}exp)$. and equations in the cryptosystem imply that this is equal to $exp(exp([b0], s.DH(p1).chosen\text{-}exp), s.DH(p2).chosen\text{-}exp)$. But the definition of $F$ says that this is equal to $t.chosen\text{-}key$, as needed.

3. $\pi = reveal(u)_a$

   This corresponds to $reveal(u)_a$ in the specification. We must show that $u \notin K$. The precondition for $reveal(u)_a$ (in $Eve$) implies that $u \in s.Eve.has$. Lemma 7.2 implies that $u \notin N$, which implies that $u \notin K$.

□

**Theorem 7.5** *$S_2(\mathcal{C}, P, A)$ implements $KD(U, P, K, A)$.*

**Proof:** By Theorem 7.4. □

# 8. Algorithms Using Structured-Key Cryptosystems

In this section, we extend the implementations of private communication and of key distribution so that they use a structured-key cryptosystem, in place of a shared key cryptosystem or base-exponent cryptosystem. For the rest of the paper, fix $\mathcal{C}$ to be any augmented structured-key cryptosystem.

## 8.1. Private Communication

### 8.1.1 Notation and assumptions

We define a shared-key cryptosystem $\mathcal{C}'$ directly from $\mathcal{C}$, by saying that $MConst_{\mathcal{C}'} = MConst_C$ and $KConst_{\mathcal{C}'} = B2_{\mathcal{C}}$. That is, we use the $B2$ terms in $\mathcal{C}$ as "names" for keys in $\mathcal{C}'$. In this subsection, we assume: $P$ is an arbitrary set with at least 2 elements; $A$ is an arbitrary set; $U = set_{\mathcal{C}}$; $M = [MConst_{\mathcal{C}}]$; $K = [B2_{\mathcal{C}}]$; $X = [XConst1_{\mathcal{C}}] \cup [XConst2_{\mathcal{C}}]$.

Also, $W$ is the set of all elements $w \in set_{\mathcal{C}}(\text{“}M\text{”})$ that can be obtained as follows: In cryptosystem $\mathcal{C}$, $w$ is obtained from an element $m \in set_{\mathcal{C}'}(\text{“}M\text{”})$ by applying some number of $enc$ operations with second arguments in $set_{\mathcal{C}}(\text{“}B''\text{”}) - K$. (That is, $w$ is obtained by "wrapping" the element.) Furthermore, $N = W \cup K \cup X$; $U' = U = set_{\mathcal{C}}$; $A'$ is an arbitrary set, disjoint from $A$.

The most interesting part of this is the definition of $W$, which is intended to designate the elements of type "$M$" that are to be avoided. Set $W$ must be sufficiently large to include all elements of type "$M$" that could help to compute elements that are supposed to remain unknown. But $W$ must be sufficiently small to exclude elements that might be communicated in other protocols with which the present protocol is going to be composed. And, $W$ must be defined reasonably simply. Coming up with a good choice of $W$ seems at this point to be something of an art, similar to coming up with a good invariant.

The choice we have made above is one of several possibilities. We think it looks a little messy, especially because it does not fall into a formalized style that might suggest how similar definitions might be made for more complicated protocols. A simpler choice that would work for this paper would be $W = set_{\mathcal{C}}(\text{“}M\text{”})$, but that seems to be ruling out more than would be ideal. Another choice would be a smaller $W$, for instance the set of elements of $U$ that are easily reachable from $M \cup (set_{\mathcal{C}}(\text{“}B''\text{”}) - K)$ in $\mathcal{C}$. We leave this for later work.

### 8.1.2 New encoder and decoder automata

The formal definitions of $Enc3$ and $Dec3$ are nearly identical to those of $Enc$ and $Dec$. The difference is that the new automata use elements of type "$B$" in place of $KConst$. Of course, the parameters have new meanings, as defined just above.

### 8.1.3 New implementation

We define $S_3$ to be the algorithm from Section 6, but implemented using the structured-key cryptosystem $\mathcal{C}$ rather than a shared key cryptosystem. That is, $S_3(\mathcal{C}, P, A, U', A')$ is the composition of the following automata, with some actions hidden:

- $Enc3(\mathcal{C}, P)_{p,q}$ and $Dec3(\mathcal{C}, P)_{p,q}$, $p, q \in P, p \neq q$.
- $IC(U, P, A)$, $Eve(\mathcal{C}, P, A)$, $Env(U, A, N)$.
- $KD(U', P, K, A')$.

To get $S_3(\mathcal{C}, P, A, U', A')$, we hide: $eavesdrop_{p,q,a}$, $p, q \in P$, $a \in A$; $IC\text{-}send_{p,q}$, $IC\text{-}receive_{p,q}$, $p, q \in P$; $grant_p$, $p \in P$; $learn_a$, $a \in A$; $reveal_a$, $a \in A'$. We want to show that this system implements $PC(U, P, M, A)$.

### 8.1.4 Invariants

**Lemma 8.1** *In all reachable states of $S_3$, the following are true:*

1. *For all $p$, $Enc^3_{p,q}.shared\text{-}key \in K \cup \{\perp\}$.*
2. *For all $p$, $Dec^3_{p,q}.shared\text{-}key \in K \cup \{\perp\}$.*
3. *For all $p, q$, if $u \in IC.buffer(p,q)$ then $u = enc(m,k)$, where $m \in M$ and $k \in K$.*
4. *For all $p, q$, all $x \in X$, $x \notin IC.buffer(p,q)$.*

**Lemma 8.2** *In all reachable states of $S_3$, the following are true:*

1. *No element of $X$ is in $Eve.has$.*
2. *Assume that $(M \cup K) \cap Eve.has = \emptyset$. If $w \in W \cap Eve.has$ and $v \in set_{\mathcal{C}'}(\text{“}M\text{”})$ is easily reachable from $\{w\} \cup (set_C(\text{“}B\text{”}) - K)$ in $\mathcal{C}$, then $v \in Eve.has$.*

Part 2 of invariant 8.2 has a somewhat different style from those we have used so far. It basically says that if a "wrapped version" of an element of $set_{\mathcal{C}'}(\text{“}M\text{”})$ is in $has$, then the actual element of $set_{\mathcal{C}'}(\text{“}M\text{”})$ must also be there. Note that we did not give any invariants here saying that $K$ elements or $M$ elements do not appear in $Eve.has$, as we did in Section 6.3. This is because (in the interests of decomposition) we are trying to avoid proving facts that have already been proved for the more abstract version of the algorithm. Instead, we are trying to rely on the simulation relation, described in the next subsection, to give such facts.

### 8.1.5 Implementation proof

We prove the correctness of $S_3$ as a consequence of that of the analogous system $S_1(\mathcal{C}', P, A, U', A')$. By our previous result about $S_1$, Theorem 6.5:

**Lemma 8.3** *$S_1(\mathcal{C}', P, A, U', A')$ implements $PC(set_{\mathcal{C}'}, P, M, A)$.*

In order to prove correctness of $S_3(\mathcal{C}, P, A, U', A')$, we would like to demonstrate a simulation relationship from $S_3(\mathcal{C}, P, A, U', A')$ to $S_1(\mathcal{C}', P, A, U', A')$. To do this, we first make the interfaces consistent, by defining $S'_3(\mathcal{C}, P, A, U', A')$ from $S_3$ by hiding the actions $reveal(u)_a$, $u \in U - set_{\mathcal{C}'}$.

**Lemma 8.4** *If $\beta$ is a trace of $S_3(\mathcal{C}, P, A, U', A')$ then $\beta$ with all $reveal(u)$ actions removed, $u \in U - set_{\mathcal{C}'}$, is a trace of $S'_3(\mathcal{C}', P, A, U', A')$.*

Now we define the relation $F$ from $S'_3(\mathcal{C}, P, A, U', A')$ to $S_1(\mathcal{C}', P, A, U', A')$: $(s,t) \in F$ provided:

1. For all components except $Eve$, all state components are identical.
2. If $u \in set_{\mathcal{C}'}$ and $u$ is easily reachable from $s.Eve.has \cup (set_{\mathcal{C}} - N)$ in $\mathcal{C}$ then $u$ is easily reachable from $t.Eve.has \cup (set_{\mathcal{C}'} - (M \cup K))$ in $\mathcal{C}'$.

**Theorem 8.5** *$F$ is a simulation relation.*

**Proof:** For the initial condition, let $s$ and $t$ be the unique start states of $S'_3(\mathcal{C}, P, A, U', A')$ and $S_1(\mathcal{C}', P, A, U', A')$, respectively. We must check that $(s,t) \in F$. The key is to show that if $u \in set_{\mathcal{C}'}$ and $u$ is easily reachable from $U - (W \cup K \cup X)$ in $\mathcal{C}$ then $u$ is easily reachable from $set_{\mathcal{C}'} - (M \cup K)$ in $\mathcal{C}'$. But properties of the cryptosystem imply that there is no such element $u$, so this is vacuously true. For the step condition, the most interesting cases are:

1. $\pi = reveal(u)_a$, $a \in A$

   We must show that $u \notin M$. So suppose for the sake of contradiction that $u \in M$. By the precondition, we know that $u \in s.Eve.has$. Since $u \in set_{\mathcal{C}'}$, the definition of $F$ implies that $u$ is easily reachable from $t.Eve.has \cup (set_{\mathcal{C}'} - (M \cup K))$ in $\mathcal{C}'$. But Lemma 6.3 implies that no element of $M$ is easily reachable from $t.Eve.has \cup (set_{\mathcal{C}'} - (M \cup K))$ in $\mathcal{C}'$. This yields the needed contradiction.

2. $\pi = compute(u, f)$

   This maps to the trivial fragment. We must argue that $(s', t) \in F$. It suffices to show that any element $u \in set_{\mathcal{C}'}$ that is easily reachable from $s'.Eve.has \cup (U - N)$ is also easily reachable from $s.Eve.has \cup (U - N)$. But this follows from general properties of the $compute$ action in $Eve$.

3. $\pi = learn(u)_a$

   We consider two cases:

   (a) $u \in set_{\mathcal{C}'}$

   Then the corresponding fragment consists of a single step, with the same action. To see that this is enabled, note that $u \notin N$, by the precondition in $S'_3$. In particular, $u \notin M \cup K$. This implies that $learn(u)$ is enabled in $S_1$.

   To show that $(s', t') \in F$, suppose that $v \in set_{\mathcal{C}'}$ and $v$ is easily reachable from $s'.Eve.has \cup (U - N)$ in $\mathcal{C}$. Then since $u \in U - N$, we have also that $v$ is easily reachable from $s.Eve.has \cup (U - N)$ in $\mathcal{C}$. Then since $(s,t) \in F$, we have that $v$ is easily reachable from $t.Eve.has \cup (set_{\mathcal{C}'} - (M \cup K))$ in $\mathcal{C}'$, which implies that $v$ is easily reachable from $t'.Eve.has \cup (set_{\mathcal{C}'} - (M \cup K))$ in $\mathcal{C}'$. This proves that $(s', t') \in F$.

(b) $u \in U - set_{\mathcal{C}'}$

Then the corresponding fragment consists of the single state $t$. We must show that $(s', t) \in F$. It suffices to show that any element $v \in set_{\mathcal{C}'}$ that is easily reachable from $s'.Eve.has \cup (U - N)$ in $\mathcal{C}$ is also easily reachable from $s.Eve.has \cup (U - N)$ in $\mathcal{C}$. But the precondition implies that $u \in U - N$, so $s'.Eve.has \cup (U - N) = s.Eve.has \cup (U - N)$, so this is obvious.

□

**Theorem 8.6** *$S_3'(\mathcal{C}, P, A, U', A')$ implements $S_1(\mathcal{C}', P, A, U', A')$.*

**Lemma 8.7** *If $\beta$ is a trace of $S_3(\mathcal{C}, P, A, U', A')$ then $\beta$ with all $reveal(u)$ actions removed, for $u \in U - set_{\mathcal{C}'}$, is a trace of $S_1(\mathcal{C}', P, A, U', A')$.*

**Proof:** By Theorem 8.6 and Lemma 8.4. □

**Theorem 8.8** *$S_3(\mathcal{C}, P, A, U', A')$ implements $PC(U, P, M, A)$.*

**Proof:** By Lemmas 8.7 and 8.3. Let $\beta$ be a trace of $S_3(\mathcal{C}, P, A, U', A')$. Then Lemma 8.7 implies that $\beta_1$ is a trace of $S_1(\mathcal{C}', P, A, U', A')$, where $\beta_1$ is equal to $\beta$ with all $reveal(u)$ actions removed, for $u \in U - set_{\mathcal{C}'}$. Then Lemma 8.3 implies that $\beta_1$ is a trace of $PC(set_{\mathcal{C}'}, P, M, A)$. It follows that $\beta_1$ is a trace of $PC(set_{\mathcal{C}}, P, M, A)$. Now, since $\beta$ differs from $\beta_1$ only by including some $reveal$ actions for elements in $U - set_{\mathcal{C}'}$, it follows that $\beta$ is a trace of $PC(set_{\mathcal{C}}, P, M, A)$. □

The proofs of the results in this and the next subsection deal with specific cryptosystems. It would be interesting to extract general theorems that could be applied to get such results. Such theorems would involve some kind of notion of "embedding" of one cryptosystem in another, and statements articulating when a protocol that works with a cryptosystem also works with any cryptosystem in which that cryptosystem is embedded.

## 8.2. Key Distribution

### 8.2.1 Notation and assumptions

We define an augmented base-exponent cryptosystem $\mathcal{C}'$ directly from $\mathcal{C}$, by saying $BConst_{\mathcal{C}'} = BConst_{\mathcal{C}}$, $XConst1_{\mathcal{C}'} = XConst1_{\mathcal{C}}$, $XConst2_{\mathcal{C}'} = XConst2_{\mathcal{C}}$, and $b0_{\mathcal{C}'} = b0_{\mathcal{C}}$. In this subsection, we assume: $P = \{p1, p2\}$; $A$ is an arbitrary set; $U = set_{\mathcal{C}}$; $K = [B2_{\mathcal{C}}]$; $X = [XConst1_{\mathcal{C}}] \cup [XConst2_{\mathcal{C}}]$; $N = K \cup X$.

### 8.2.2 New implementation

The new endpoint automata are syntactically the same as the old endpoint automata. The only difference is that the subscript $\mathcal{C}$ now refers to a structured-key cryptosystem. We define $S_4$ to be the algorithm from Section 7, but implemented using the structured-key cryptosystem $\mathcal{C}$ rather than a base-exponent cryptosystem. That is, $S_4(\mathcal{C}, P, A)$ is the composition of the following automata, with some actions hidden:

- $DH(\mathcal{C}, P)_p$, $p \in P$.
- $IC(U, P, A)$, $Eve(\mathcal{C}, P, A)$, $Env(U, A, N)$.

To get $S_4(\mathcal{C}, P, A)$, we hide: $eavesdrop_{p,q,a}$, $p, q \in P$, $a \in A$; $IC\text{-}send_{p,q}$, $IC\text{-}receive_{p,q}$, $p, q \in P$; $learn_a$, $a \in A$. We want to show that this system implements $KD(U, P, K, A)$.

We prove the correctness of $S_4$ as a consequence of that of the analogous system $S_2(\mathcal{C}', P, A)$. By our previous result about $S_2$, Theorem 7.5:

**Lemma 8.9** *$S_2(\mathcal{C}', P, A)$ implements $KD(set_{\mathcal{C}'}, P, K, A)$.*

**Lemma 8.10** *If $\beta$ is a trace of $S_4(\mathcal{C}, P, A)$ then $\beta$ with all $reveal(u)$ actions removed, for $u \in U - set_{\mathcal{C}'}$, is a trace of $S_4'(\mathcal{C}', P, A)$.*

Now we define the relation $F$ from $S_4'(\mathcal{C}, P, A)$ to $S_2(\mathcal{C}', P, A)$: $(s, t) \in F$ provided:

1. For all components except $Eve$, all state components are identical.
2. If $u \in set_{\mathcal{C}'}$ and $u$ is easily reachable from $s.Eve.has \cup (set_{\mathcal{C}} - N)$ in $\mathcal{C}$ then $u$ is easily reachable from $t.Eve.has \cup (set_{\mathcal{C}'} - N)$ in $\mathcal{C}'$.

**Theorem 8.11** *$F$ is a simulation relation.*

**Proof:** Analogous to that of Theorem 8.5. □

**Theorem 8.12** *$S_4'(\mathcal{C}, P, A)$ implements $S_2(\mathcal{C}', P, A)$.*

**Lemma 8.13** *If $\beta$ is a trace of $S_4(\mathcal{C}, P, A)$ then $\beta$ with all $reveal(u)$ actions removed, for $u \in U - set_{\mathcal{C}'}$, is a trace of $S_2(\mathcal{C}', P, A)$.*

**Theorem 8.14** *$S_4(\mathcal{C}, P, A)$ implements $KD(U, P, K, A)$.*

# 9. Putting the Pieces Together

Now we describe how to put the previous results together, to get an implementation of private communication that uses the shared-key communication protocol in combination with the Diffie-Hellman key distribution service. The first step combines the two protocols, but still keeps the insecure channels, eavesdroppers, and environments for the two algorithms separate. The second step combine the two channels into one and likewise for the eavesdroppers and the environments.

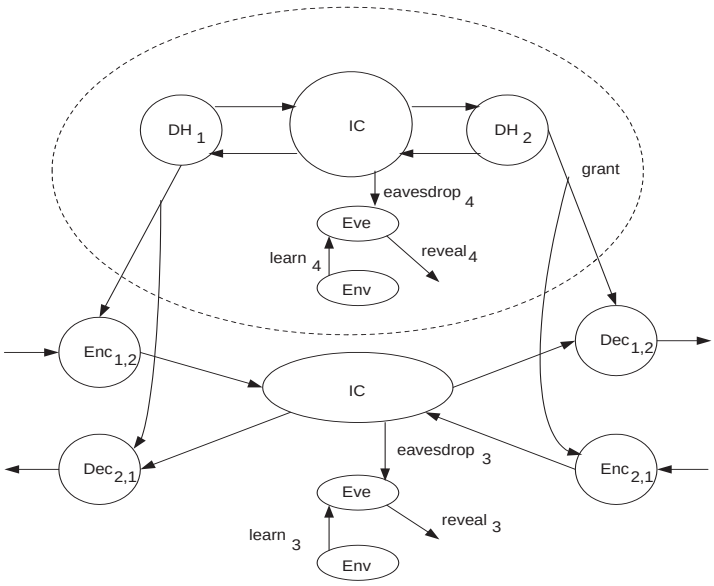


**Figure 3.** $S_5$

### 9.1. Composing Diffie-Hellman and Shared-Key Communication to get Private Communication

Recall that we have already fixed $\mathcal{C}$ to be an augmented structured-key cryptosystem. We now fix, for the rest of the paper: $U = set_{\mathcal{C}}$; $P = \{p1, p2\}$; $P' = \{p1', p2'\}$; $A$ is an arbitrary set; $M = [MConst_{\mathcal{C}}]$; $K = [B2_{\mathcal{C}}]$; $X = [XConst1_{\mathcal{C}}] \cup [XConst2_{\mathcal{C}}]$; $W$ is the set of elements of $set_{\mathcal{C}}$("$M$") that can be obtained from $set_{\mathcal{C}'}$("$M$") $\cup$ $(set_{\mathcal{C}}$("$B''$") $- K)$ in $\mathcal{C}$ using $enc$; $N = W \cup K \cup X$; $A'$ is an arbitrary set, disjoint from $A$. The combined system $S_5$ consists of the following pieces:

- $Enc3(\mathcal{C}, P)_{p,q}$, $Dec3(\mathcal{C}, P)_{p,q}$, $p, q \in P, p \neq q$.
- $DH5_p$, $p \in P$; each of these is a renamed version of $DH(\mathcal{C}, P)_p$, with the subscripts in $IC\text{-}send_{p,q}$ and $IC\text{-}receive_{q,p}$ actions renamed to their primed versions.
- $IC(U, P, A)$, $Eve(\mathcal{C}, P, A)$, $Env(U, A, N)$.
- $IC(U, P', A')$, $Eve(\mathcal{C}, P', A')$, $Env(U, A', N')$.

$S_5$ hides all the actions except for the $PC\text{-}send$, $PC\text{-}receive$, and $reveal_a$ actions for $a \in A$.

Figure 3 contains an interaction diagram for $S_5$.

**Theorem 9.1** *$S_5$ implements $PC(U, P, M, A)$.*

**Proof:** This follows from Theorems 8.14 and 8.8, using general projection and pasting lemmas for I/O automata. ∎

### 9.2. Merging Channels, Adversaries, and Environments

The final implementation, $S_6$, is obtained from $S_5$ by merging the two separate insecure channels into one, and likewise for the two adversaries and the two environments. To do this, and yet keep the same interfaces, we extend the definitions of $IC$ and $Eve$ to allow two types of ports, primed and unprimed. Specifically, $S_6$ consists of:

- $Enc3(\mathcal{C}, P)_{p,q}$, $Dec3(\mathcal{C}, P)_{p,q}$, $p, q \in P, p \neq q$.
- $DH5_p$, $p \in P$.
- $IC(U, P, A, P', A')$.
- $Eve(\mathcal{C}, P, A, P', A')$.
- $Env(U, A \cup A', N \cup N')$.

Here, the extended $IC$ is the same as $IC(U, P \cup P', A \cup A')$ but only has actions with subscripts $p, q, a$ where either $p, q \in P$, $a \in A$ or $p, q \in P'$, $a \in A'$. Similarly for the extended $Eve$. Also, $S_6$ hides all actions except for the $PC\text{-}send$, $PC\text{-}receive$, and $reveal_a$ actions for $a \in A$.

The combined eavesdropper eavesdrops and learns on all adversary ports in $A \cup A'$, and can use all this information in calculating its $has$ information, which resides in a single state component. The combined environment avoids communicating any information in $N \cup N'$. We claim that $S_6$ implements $S_5$, which implies that $S_6$ implements $PC(U, P, M, A)$. To prove this result, we define $S_7$, which is just like $S_5$ except that it combines the eavesdroppers, but not the channels or environments.

**Lemma 9.2** *$S_7$ implements $S_5$.*

The essence of this lemma is that information that an eavesdropper can acquire in either protocol does not upset the requirements of the other protocol. It would be nice to show that $Eve(\mathcal{C}, P, A, P', A')$ implements $Eve(\mathcal{C}, P, A) \times Eve(\mathcal{C}, P', A')$. But this is not quite correct: the implementation relationship requires assumptions about the contexts in which the eavesdroppers run, (and also, the relationship does not preserve the $learn$ actions). So we just prove an implementation relationship for the eavesdroppers in their contexts. However, it is clear that the argument uses only minimal information about the particular contexts, namely, what kind of information they can contribute in $eavesdrop$ and $learn$ actions; it should be possible to extract a general lemma stating these restrictions on contexts explicitly.

**Proof:** We use a simulation relation $F$ from $S_7$ to $S_5$ defined so that $(s, t) \in F$ exactly if:

1. Everything except for $has$ components is the same in $s$ and $t$.

2. If $u \in s.has$ then $u$ is easily reachable from $t.Eve(\mathcal{C}, P, A).has \cup (U - N)$ in $\mathcal{C}$.

3. If $u \in s.has$ then $u$ is easily reachable from $t.Eve(\mathcal{C}, P', A').has \cup (U - N')$ in $\mathcal{C}$.

This mapping says, essentially, that any information that the combined eavesdropper acquires is something that either of the individual eavesdroppers could have acquired anyway. We show that this is a simulation. The initial conditions are immediate, because $s.has$ is empty. We consider steps:

1. $reveal(u)_a, a \in A$

   We know that $u \in s.has$. So by definition of $F$, we have that $u$ is easily reachable from $t.Eve(\mathcal{C}, P, A).has \cup (U - N)$. Let this step correspond to a sequence of $learn_a$ and $compute_a$ actions sufficient to put $u$ into $Eve(\mathcal{C}, P, A).has$, followed by the same $reveal(u)_a$ action. The sequence of learns and computes guarantees that the action is enabled in the spec.

2. $reveal(u)_a, a \in A'$

   Analogous to the previous case.

3. $learn(u)_a, a \in A \cup A'$

   Map this to the trivial execution fragment. By the precondition, this adds something in $U - (N \cup N')$ to $s.has$. But this is obviously easily reachable from $t.Eve(\mathcal{C}, P, A).has \cup (U - N)$ and from $t.Eve(\mathcal{C}, P', A').has \cup (U - N')$. So the correspondence is preserved.

4. $compute(u, f)_a, a \in A \cup A'$

   Map to the trivial fragment. The precondition implies that $u$ is easily reachable from $s.has$. By the inductive step, all the elements of $s.has$ that are needed for this computation are easily reachable from $t.Eve(\mathcal{C}, P, A).has \cup (U - N)$ and from $t.Eve(\mathcal{C}, P', A').has \cup (U - N')$. So $u$ is also easily reachable from $t.Eve(\mathcal{C}, P, A).has \cup (U - N)$ and from $t.Eve(\mathcal{C}, P', A').has \cup (U - N')$ (using one more step).

5. $eavesdrop(u)_a, a \in A$

   Then Lemma 8.1 implies that $u$ is of the form $enc(m, k)$, $m \in M$, $k \in K$. Therefore, $u \in U - N'$. Let this correspond to the same action in the spec. We must show that $(s', t') \in F$. This means we must show that $u$ is easily reachable from $t'.Eve(\mathcal{C}, P, A).has \cup (U - N)$ in $\mathcal{C}$ and $u$ is easily reachable from $t'.Eve(\mathcal{C}, P', A').has \cup (U - N')$ in $\mathcal{C}$.

   By the effect of the action in the spec, $u \in t'.Eve(\mathcal{C}, P, A).has$, so $u$ is obviously easily reachable from $t'.Eve(\mathcal{C}, P, A).has \cup (U - N)$ in $\mathcal{C}$. Also, since $u \in U - N'$, $u$ is obviously easily reachable from $t'.Eve(\mathcal{C}, P', A').has \cup (U - N')$ in $\mathcal{C}$.

6. $eavesdrop(u)_a, a \in A'$

   Then $u$ is of the form $exp(b0, x) \in U - N$ Let this correspond to the same action in the spec. We must show that $(s', t') \in F$. This means we must show that $u$ is easily reachable from $t'.Eve(\mathcal{C}, P, A).has \cup (U - N)$ in $\mathcal{C}$ and $u$ is easily reachable from $t'.Eve(\mathcal{C}, P', A').has \cup (U - N')$ in $\mathcal{C}$.

   By the effect of the action in the spec, $u \in t'.Eve(\mathcal{C}, P', A').has$, so $u$ is obviously easily reachable from $t'.Eve(\mathcal{C}, P', A').has \cup (U - N')$ in $\mathcal{C}$. Since $u \in U - N$, $u$ is obviously easily reachable from $t'.Eve(\mathcal{C}, P, A).has \cup (U - N)$ in $\mathcal{C}$.

□

**Lemma 9.3** *$S_6$ implements $S_7$.*

The proof of Lemma 9.3 is easy, based on the following two lemmas:

**Lemma 9.4** *$Env(U, A \cup A', N \cup N')$ implements $Env(U, A, N) \times Env(U, A', N')$.*

**Lemma 9.5** *$IC(U, P, A, P', A')$ implements $IC(U, P, A) \times IC(U, P', A')$.*

This all yields:

**Lemma 9.6** *$S_6$ implements $S_5$.*

**Proof:** By Lemmas 9.3 and 9.2. □

**Theorem 9.7** *$S_6$ implements $PC(U, P, M, A)$.*

**Proof:** By Lemmas 9.6 and Theorem 9.1. □

## 10. Discussion

In this paper, we have modeled and analyzed the combination of simple shared key communication with Diffie-Hellman key distribution, in the presence of an eavesdropper adversary. Although this example is very simple, many kinds of decomposition are evident in its presentation. Understanding these in a simple context is a prerequisite for extending them to more complicated protocols.

We believe that this type of presentation is useful in clarifying protocol issues. It also helps in separating the protocol issues from other issues, such as cryptosystem reachability issues, which can be treated separately. It appears possible

to decompose the presentation in this paper even more, for example, by defining a notion of embeddings of cryptosystems and obtaining the results of Section 8 as consequences of such theorems.

In work in progress, we are extending these ideas to more complex protocols like that of Diffie, Oorschot, and Weiner [5], which tolerate more active adversaries. So far, it appears that the modeling/analysis ideas of this paper scale well to the more complicated examples. Some issues that arise in modeling the protocol of [5] are: The cryptosystems are more complicated, so more complicated arguments need to be made about reachability; for example, the analogues of the set $W$ defined in Section 8.1.1 become more complicated. Also, because the adversary has more active control of the communication system, it is convenient to combine them into a single automaton model. (The $has$ component of that automaton is now used to decide what may be delivered to the client, as well as what may be revealed.) Also, the correctness guarantees are weaker—for instance, repeated deliveries of the same message, and deliveries to the wrong recipient, are possible. A more complicated key distribution service specification will also be needed, including key requests and granting of multiple keys.

The work of this paper has not mentioned liveness properties. For the simple case of this paper, with a passive eavesdropper, liveness claims are certainly possible. They can be incorporated easily into the model in the form of time bounds, and proved using the usual assertional methods for timing analysis, such as those appearing in [3, 11]. For more active adversaries, more sophisticated algorithms can also guarantee liveness properties, which could also be formulated as time bounds and proved similarly.

Another interesting research direction is the modular introduction of probabilistic considerations. We expect that it is possible to accomplish a great deal at a high level of abstraction, by simply assuming that certain low probability "bad" events do not occur. The low probability bad events could then be introduced separately, with general theorems used to limit their impact on system behavior. But such general theorems remain to be developed.

## References


[1] M. Abadi. Protection in programming-language translations. In *Automata, Languages and Programming: 25th International Colloquium (ICALP'98)*, pages 868–883, July 1998. Also, Digital SRC Research Report 154 (April 1998), Palo Alto, CA.

[2] M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 105–116, June 1998.

[3] H. Attiya and N. A. Lynch. Time bounds for real-time process control in the presence of timing uncertainty. *Information and Computation*, 110(1):183–232, April 1994.

[4] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–656, November 1976.

[5] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentification and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[6] A. Fekete, M. F. Kaashoek, and N. Lynch. Implementing sequentially consistent shared objects using broadcast and point-to-point communication. *Journal of the ACM*, 45(1):35–69, January 1998.

[7] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partitionable group communication service. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 53–62, Santa Barbara, CA, August 1997. Expanded version in [8].

[8] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partitionable group communication service. Technical Memo MIT-LCS-TM-570, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1997. Also, submitted for journal publication.

[9] J. Hickey, N. Lynch, and R. van Renesse. Specifications and proofs for Ensemble layers. In R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems* (Fifth International Conference, TACAS'99, Amsterdam, the Netherlands, March 1999, volume 1579 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 1999.

[10] B. Lampson and A. Shvartsman. POCS Course Notes (Principles of Computer Systems), 1997. Available online at ftp://theory.lcs.mit.edu/pub/classes/6.826/www/6.826-top.html.

[11] V. Luchangco. Using simulation techniques to prove timing properties. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, June 1995.

[12] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, March 1996.

[13] N. Lynch and S. Rajsbaum. On the Borowsky-Gafni simulation algorithm. In *Proceedings of the Fourth ISTCS: Israel Symposium on Theory of Computing and Systems*, pages 4–15, Jerusalem, Israel, June 1996. IEEE Computer Society. Also, short version appears in *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, Philadelphia, PA, page 57, May 1996.

[14] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, September 1989. Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands. Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, November 1988.

[15] L. C. Paulson. Inductive approach to verifying cryptographic protocols. *J. Comp. Security*, 1998.

[16] T. P. Petrov, A. Pogosyants, S. J. Garland, V. Luchangco, and N. A. Lynch. Computer-assisted verification of an algorithm for concurrent timestamps. In R. Gotzhein and J. Bredereke, editors, *Formal Description Techniques IX: Theory, Applications, and Tools* (FORTE/PSTV'96: Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification, Kaiserslautern, Germany, October 1996), pages 29–44. Chapman & Hall, 1996.

[17] M. Vaziri. Naming state variables of composite automata in IOA. Manuscript, Nov. 9, 1998.