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

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

Technical Report MIT/LCS/TR-789
MIT Laboratory for Computer Science

August 9, 1999

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. Ways of decomposing the reasoning task into clearly separable pieces are needed. 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* [26, 23] 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., [23, 32, 25]), and about several distributed system designs (see, e.g., [13, 14, 17, 20]), 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 [10] 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-van Oorschot-Weiner key distribution protocol [11], 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, understanding how 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 [30] or the strand space techniques of Fabrega, et al. []. However, a general framework should provide a rigorous way of combining the different types of issues.

Similarly, we try to decompose the distributed algorithms themselves as much as possible, by:

1. Treating sub-protocols separately, then combining them using general theorems about automaton composition.

2. Giving very high level automaton specifications for services, giving separate, detailed descriptions of implementing algorithms, and showing, by means of simulation relations, that the algorithms implement the services.
3. 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.
4. 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., [23], 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 include in traces special *fail* input actions that signal the occurrence of failure events (see, e.g., [23, 25]). Sometimes it is convenient to consider different strengths of failure actions (e.g., the *good*, *bad*, and *ugly* failure actions in [14]). Also, in order to treat timing properties compositionally, it is useful to include timing information into traces.

In the case of security protocols, some 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 presents a model for cryptosystems, which describe the data types encountered in the protocols, including 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. The data model is similar to others in the literature. Section 3 contains a brief review of the I/O automaton model. Section 4 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, for example, 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 together 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.

Related work:

The I/O automaton model is similar to the labeled transition system models underlying process algebras. However, notation and proof techniques typically used for I/O automata differ greatly from the usual process algebraic notations and methods; notably, work based on I/O automata uses explicit, structured representations of automaton states.

Many researchers have stated and proved invariant assertions for security protocols (see, for example, [19, 36, 33, 30]). On the other hand, simulation relations have not been used much in prior work on reasoning about security protocols. An example of work using simulation relation ideas is the work on "safe simplifying transformations" by Hui and Lowe [18]. Also, Abadi and co-workers have used simulation relation notions in proving equivalences for components of secure systems (see, e.g., [1, 3]).

Our strategy of including in traces explicit information about what can be learned and what can be revealed is a key to our approach to compositional reasoning about security protocols. Including this information makes simple traces rich enough to express at least some interesting security properties. A similar strategy, called "negative constraints", is used by Cavalca and Segala in analyzing authentication protocols [8, 9]. This strategy differs from the "zero-knowledge" approach to proving secrecy properties (e.g., [16]) by specifying the particular information that can be learned and revealed, rather than assuming that nothing is learned or revealed. This extra flexibility makes it easier to compose specifications. Another difference between our work and work on zero-knowledge is that zero-knowledge proofs include probabilistic considerations, which we have so far avoided.

Bellare and Rogaway have developed a framework for composing security protocols [6, 7]. Their approach is less formal than ours, but it takes probabilities into account. Lincoln, Mitchell, Mitchell, and Scedrov [21] present a formal approach to studying the interactions between protocols and cryptographic primitives, again taking probabilities into account. This work can be regarded as a more formal version of the work of Bellare and Rogaway. It is based on a form of π-calculus [29] and probabilistic polynomial time process models.

Our work differs from work on formal logics for security for example, the BAN logic of Burrows, Abadi, and Needham [2], in that ours is carried out entirely at the level of automaton semantics. However, our work is compatible with work on security logics, in that it should be possible to express our proof methods using formal logics. Also, it should be possible to interpret some security logics in terms of I/O automata; the effort would be similar to Abadi and Tuttle's construction of an automaton semantics for a derivative of the BAN logic [4].

Our work makes extensive use of inductive proofs, mainly for verifying invariants and simulation relations. Paulson [30] has developed an extensive collection of methods for reasoning inductively about cryptographic protocols, all supported by the Isabelle interactive theorem prover [31]. His approach includes some methods for proving secrecy properties, which involve showing that certain values are not reachable from other values within cryptosystems. We think that such methods may be useful for constructing formal proofs of cryptosystem unreachability results like the ones needed in this paper. Other approaches that should be useful in proving cryptosystem reachability results include the *rank function* approach of Schneider [33] and the *strand space* techniques of Fabrega et al.

All of the related work mentioned above adopts a model where the adversary may be active, not just an eavesdropper. We believe that this is not a fundamental difference, in that our general approach can be extended to model more active adversaries.

Sheyner and Wing [34] have formalized much of the approach of this paper using conservative extensions to theories supplied with the theorem prover Isabelle. In particular, they have formalized shared-key cryptosystems, private communication, and essentially all the automata appearing in Section 6 of this paper. They have carried out interactive proofs using Isabelle for the fact that S_1 simulates PC, for the invariants in Section 6.3, and for several other invariants useful in the simulation argument. They are continuing to model other security protocols using the same approach.

An earlier version of the present paper appeared in the 12th IEEE Computer Security Foundations Workshop [24].[1]

2 Data Model

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

2.1 Cryptosystems

We use λ to denote the empty string.
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:

[1] Unfortunately, subsection numbers are messed up in that version. A corrected copy of that paper appears at URL `http://theory.lcs.mit.edu/tds/papers/Lynch/CSFW.html`.

- 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 $fun_{\mathcal{C}}(f) : set_{\mathcal{C}}(t_1) \times \cdots \times set_{\mathcal{C}}(t_k) \to set_{\mathcal{C}}(t)$.

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 $EN_{\mathcal{C}}$.

2.2 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 $e \in E$.

2.3 Cryptosystem Examples

In this subsection we give the specific kinds of cryptosystems used later in this paper. These are: shared-key cryptosystems, used in shared-key communication; base-exponent cryptosystems, used in Diffie-Hellman key distribution; and structured-key cryptosystems, which are essentially combination of shared-key and base-exponent cryptosystems, and are used when shared-key communication and Diffie-Hellman key distribution protocols are combined.

2.3.1 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 define R to be the smallest congruence relation on $Terms_{\mathcal{S}}$ that groups together all terms that are related by the given equations.

The following lemma gives some basic properties of a shared-key cryptosystem, used later in the proof of an invariant for a shared-key communication protocol (Lemma 6.3). Many properties of this sort are needed in this paper. However, we will not continue to be as explicit as we are here, but will revert to simply citing "properties of the cryptosystem". A careful treatment of such properties is a separate effort, and would benefit from the use of other methods, as discussed in the Introduction.

Lemma 2.1 *Let $\mathcal{C}$ be a shared-key cryptosystem, $\mathcal{S}$ its signature and R its congruence relation.*

1. *Suppose that e_1 and e_2 are terms of type "M" with $e_1 R e_2$. Let enc_i and dec_i denote the respective number of occurrences of enc and dec in e_i, $i \in \{1, 2\}$.*
   Then $enc_1 - dec_1 = enc_2 - dec_2$.
2. *For all $m_1, m_2 \in MConst_{\mathcal{S}}$, $k \in KConst_{\mathcal{S}}$:*
   $enc(m_1, k)$ is not R-related to m_2.

Proof: Part 1 is proved by induction on the number of substitutions required to relate one term to the other. Since there is only one kind of substitution, and it preserves this difference, the result holds. Part 2 follows from Part 1. ∎

2.3.2 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, and $FN_{\mathcal{S}}$ consists of:

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

$EN_{\mathcal{S}} = \{exp\} \cup BConsts_{\mathcal{S}}$. 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 $b \in BConsts_{\mathcal{S}}$, $x \in XConst1_{\mathcal{S}}$ 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 $BConsts_{\mathcal{S}}$.

2.3.3 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", and $FN_{\mathcal{S}}$ 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 $range(exp) = \text{"}B\text{"}$.
- $MConsts_{\mathcal{S}}$, a set of message constant names, with $range(m) = \text{"}M\text{"}$ for all $m \in MConsts_{\mathcal{S}}$.
- $BConsts_{\mathcal{S}}$, 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 $x \in XConst1_{\mathcal{S}} \cup XConst2_{\mathcal{S}}$.

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

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

3 Input/Output Automata

We use I/O automata as defined in [23]. Briefly, an I/O automaton A is a state machine having a *signature* consisting of a set of *actions*, classified as *input*, *output*, and *internal* actions. A also has a set of *transitions*, which are (state, action, state) triples. It is assumed that every input action is enabled in every state. Since we do not deal with liveness in this paper, the *tasks* defined in [23] are irrelevant.

An *execution fragment* of A is an alternating (state, action, state,...) sequence, where successive triples correspond to transitions of A. An *execution* is an execution fragment that begins with a start state. The external behavior of A is modelled by the set of *traces*, which are the sequences of external actions arising from the executions.

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. Parallel composition of automata is defined by identifying external actions with the same name in different automata. We use notions of invariants and simulation relations in the usual ways; for definitions, see, for example [23].

In particular, a *simulation relation* from A to B is a relation F from $states(A)$ to $states(B)$ satisfying the following two properties:

1. Each start state of A is F-related to some start state of B.

2. For each step (s_A, π, s'_A) of A and each state s_B of B with $(s_A, s_B) \in F$, there is a "corresponding" execution fragment of B: it has the same trace as the given step, and spans from s_B to some state s'_B, where $(s'_A, s'_B) \in F$.

The key fact about a simulation relation is expressed by:

Theorem 3.1 *If there is a simulation relation from A to B then A implements B.*

4 Some Generally-Useful Automata

In this section, we give automaton models for some system components that will be used frequently in modeling security protocols, namely, 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, for example, by using trace properties) for uniformity with the way we will model algorithms and system specifications, and because this makes it possible to reason about them assertionally.

4.1 Environment Automata

In this subsection we assume that U is a universal set of data values, A is a nonempty finite set of adversary ports (that is, locations where information can be communicated to an 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. The specification says that the environment is theoretically 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:
None

Output:
$learn(u)_a$, $u \in U$, $a \in A$

States:

No variables

Transitions:

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

4.2 Insecure Channel Automata

In this subsection we assume that U is a universal set of data values, P is a nonempty finite set of client ports, and A is a nonempty finite set of adversary ports. The insecure channel admits *send* and *receive* actions for all elements of U. It 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

In this subsection we assume that $\mathcal{C}$ is a cryptosystem, P is a nonempty finite set of client ports, and A is a nonempty 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: When $eavesdrop(u)_{p,q,a}$ or $learn(u)_a$ occurs, u gets added to has. Also, 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". Similar treatments of known information appear elsewhere in the literature, for example, in [12, 19, 28, 27].

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

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., [22, 23]).

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 at any "adversary port". 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 a nonempty finite set of client ports, $M \subseteq U$ is a set of messages, and A is a nonempty finite set of adversary ports. This specification makes no mention of 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

Properties 1 and 2 above, which express at-most-once delivery of messages that were actually sent, are expressed by the transition definitions for *PC-send* and *PC-receive*. Property 3, secrecy, 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 service does not grant any other values, and does not reveal any key in K at any adversary port. The simplified key distribution service is specified by the automaton $KD(U, P, K, A)$, where U is a universal set of data values, P is a nonempty finite set of client ports, $K \subseteq U$ is a set of keys, and A is a nonempty 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-key

States:

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

Transitions:

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

$grant(u)_p$
  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 $p, q \in P$, $p \neq q$. The encoder encrypts messages from client p using the granted key, and sends the encrypted messages on the insecure channel from p to q. Note that, in the code for *IC-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*

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

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

The decoder receives messages from the insecure channel from p to q, decrypts them, and delivers the decrypted messages to q.

$Dec(\mathcal{C}, P)_{p,q}$**, 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 $set_{\mathcal{C}}(\text{“}M\text{”})$, initially empty
$shared\text{-}key \in [KConst_{\mathcal{C}}] \cup \{\perp\}$, initially $\perp$

Transitions:

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

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

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

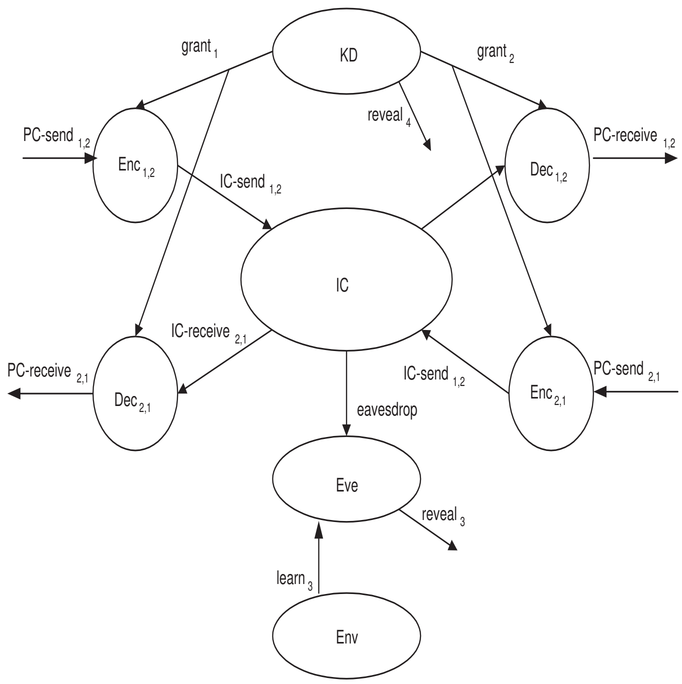

Figure 1: S_1; $P = \{1, 2\}$, $A = \{3\}$; $A' = \{4\}$

6.2 The Complete Implementation

In the rest Section 6, we assume that $U = set_{\mathcal{C}}$, $M = [MConst_{\mathcal{C}}]$, $K = [KConst_{\mathcal{C}}]$, $N = M \cup K$, U' is a set with $K \subseteq U'$, and A' is a nonempty finite set, disjoint from A.

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 constructed by composing the following automata:

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

and then hiding all the *eavesdrop*, *IC-send*, *IC-receive*, *grant*, and *learn* actions, and all the $reveal_a$ actions for $a \in A'$. That is, we hide all but the external actions of $PC(U, P, M, A)$, which are the *PC-send* and *PC-receive* actions, and the $reveal_a$ actions for $a \in A$. We sometimes omit explicit mention of parameters of S_1 (and of other systems and components), when we think that confusion is unlikely. Figure 1 contains an interaction diagram for S_1.

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

Our system model says that the eavesdropper learns no elements of $N = M \cup K$ from outside sources. This choice of N is fine for this protocol, but we do not have a general prescription for how to choose useful sets N for all protocols. "Useful" 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. The work of coming up with a good choice of N seems to be something of an art, similar to coming up with a useful invariant.

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 [35]. 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 $Dec_{p,q}.shared\text{-}key = KD.chosen\text{-}key$.*

Proof: By a simple induction on the length of an execution leading to a state. ∎

The next invariant says that all elements that appear in the insecure channel are of type "M".

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

1. *If u is in $IC.buffer_{p,q}$ then $u \in set_{\mathcal{C}}(\text{"}M\text{"})$.*

The next invariant says that no element of N ($= M \cup K$; recall that $M = [MConst_{\mathcal{C}}]$) appears in the insecure channel.

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

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

Proof: By induction on the length of an execution.
Base: The claim is true in the initial state, because the channel is initially empty.
Inductive step: Consider a step (s, π, s') of the implementation, where s satisfies the invariant. The interesting case is:

1. $IC\text{-}send(u)_{p,q}$, where $u = enc(m,k)$

   The precondition and type considerations imply that $m \in [MConst_{\mathcal{C}}]$ and $k \in [KConst_{\mathcal{C}}]$. So $m \cap MConst_{\mathcal{C}} \neq \emptyset$; let m' be any element in $m \cap MConst_{\mathcal{C}}$. Similarly, $k \cap KConst_{\mathcal{C}} \neq \emptyset$; let k' be any element in $k \cap KConst_{\mathcal{C}}$. Then $enc(m', k') \in u$.

   We claim that $u \notin [MConst_{\mathcal{C}}]$. Suppose it is and let m'' be any element in $u \cap MConst_{\mathcal{C}}$. Then $[enc(m', k')] = u = [m'']$. But Lemma 2.1 implies that $enc(m', k')$ and m'' are not equivalent terms. It follows that $u \notin [MConst_{\mathcal{C}}]$, which implies that this event does not add an element of $M = [MConst_{\mathcal{C}}]$ to the channel IC.

   The fact that this event does not add an element of $K = [KConst_{\mathcal{C}}]$ to the channel is easy to see, because the *type* of any element in any equivalence class in $[KConst_{\mathcal{C}}]$ is "K", the *type* of any element in $enc(m,k)$ is "M", and elements of one equivalence class all have the same type.

□

As a corollary to the previous invariant, we can show that no N elements appear in *Eve.has*.

Lemma 6.4 *In all reachable states of S_1, the following are true:*

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

Proof: By induction.
Base: The claim is true in the initial state, because the *Eve.has* is initially empty.
Inductive step: Consider a step (s, π, s') of the implementation, where s satisfies the invariant. The interesting cases are:

1. $\pi = eavesdrop(u)_{p,q,a}$

   The fact that this preserves the claim follows from Lemma 6.3, applied to state s.

2. $\pi = learn(u)_a$, $a \in A$

   The precondition (in $Env(U, A, N)$) says that $u \notin N$, so this cannot cause a violation.

3. $\pi = compute(u, f)_a$, $a \in A$

   Since the claim is true in s, it must be that no element of N is in $s.Eve.has$. But this means that no equivalence class of type "K" is included in $s.Eve.has$ (because the only such classes are the ones in $[KConst_{\mathcal{C}}]$). But then π cannot be enabled, because both easy functions, *enc* and *dec*, depend on a key class being in *has*.

□

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 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 6.5 *F is a simulation relation.*

Proof: We check the two conditions required in the definition of a simulation relation:
Start condition: This is easy, because all the relevant multisets are empty.
Step condition: Consider (s, π, s') in the implementation, and $(s, t) \in F$, where both s and t are reachable states. The interesting cases are:

1. $\pi = \mathit{IC\text{-}send}(u)_{p,q}$, where $u = enc(m, k)$

   This maps to the trivial one-state execution fragment t of $PC(U, P, M, A)$. We must argue that $(s', t) \in F$. This follows because this event removes m from $Enc_{p,q}.\mathit{buffer}$ as it adds the encoded version u to the insecure channel, and because of the equations relating *enc* and *dec*. Lemma 6.1 is also used here, to ensure that the keys used for encoding and decoding are the same.

2. $\pi = \mathit{IC\text{-}receive}(u)_{p,q}$

   The key point is that u is accepted by *Dec*, because it is of type "M". This following from Lemma 6.2.

3. $\pi = \mathit{PC\text{-}receive}(u)_{p,q}$

   This corresponds to the same action in the specification automaton. In this step, $u = dec(m, s.KD.\mathit{chosen\text{-}key})$ for some $m \in s.Dec_{p,q}.\mathit{buffer}$ (we use Lemma 6.1 here). Thus, by definition of the correspondence F, $u \in t.\mathit{buffer}(p, q)$, which means that π is enabled in the specification automaton, in state t. Let t' be the unique resulting state.

   To show that $(s', t') \in F$, the key facts are that one copy of m is removed from $s.Dec_{p,q}.\mathit{buffer}$ while a copy of u is removed from the abstract channel $t.\mathit{buffer}(p, q)$. Since $u = dec(m, s.KD.\mathit{chosen\text{-}key})$, this preserves the correspondence between the multisets.

4. $\pi = \mathit{reveal}(u)_a$

   This corresponds to $\mathit{reveal}(u)_a$ in the specification. We must show that $u \notin M$. The precondition for $\mathit{reveal}(u)_a$ (in *Eve*) implies that $u \in s.Eve.has$. Lemma 6.4 implies that $u \notin N$, which implies that $u \notin M$.

□

Theorem 6.6 *$S_1(\mathcal{C}, P, A, U', A')$ implements $PC(U, P, M, A)$.*

Proof: By Theorem 6.5 and Theorem 3.1. □

(An expanded version of) the results of this section have been checked by Sheyner and Wing using the Isabelle theorem prover.

7 Diffie-Hellman Key Distribution Protocol

This section describes the Diffie-Hellman key distribution protocol. Throughout the section, we assume that $\mathcal{C}$ 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. The automaton for $p1$ chooses an exponent x from the set *XConst1*, raises the distinguished base element $b0$ to the power x, and sends the result to $p2$. When it receives a corresponding value from $p2$, it raises that value to the power x and grants the result to the client as a key.

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

Signature:

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

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

Internal:
$\quad$ $\textit{choose-exp}_{p1}$

States:

$\textit{chosen-exp} \in [\textit{XConst1}_{\mathcal{C}}] \cup \{\perp\}$, initially $\perp$
$\textit{base-sent}$, a Boolean, initially *false*
$\textit{rcvd-base} \in set_{\mathcal{C}}(\text{“}B\text{”}) \cup \{\perp\}$, initially $\perp$
granted, a Boolean, initially *false*

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

Transitions:

$\textit{choose-exp}_{p1}$
$\quad$ Precondition:
$\quad\quad$ $\textit{chosen-exp} = \perp$
$\quad$ Effect:
$\quad\quad$ $\textit{chosen-exp} :=$ choose x
$\quad\quad\quad$ where $x \in [\textit{XConst1}_{\mathcal{C}}]$

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

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

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

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

$DH(\mathcal{C}, P)_{p2}$:

Signature:

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

Internal:
  $choose\text{-}exp_{p2}$

States:

$chosen\text{-}exp \in [XConst2_{\mathcal{C}}] \cup \{\perp\}$, 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:

$choose\text{-}exp_{p2}$
  Precondition:
    $chosen\text{-}exp = \perp$
  Effect:
    $chosen\text{-}exp :=$ choose x
      where $x \in [XConst2_{\mathcal{C}}]$

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

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

$grant(b)_{p2}$
  Precondition:
    $chosen\text{-}exp \neq \perp$
    $rcvd\text{-}base \neq \perp$
    $b = exp(rcvd\text{-}base, chosen\text{-}exp)$
    $granted = false$
  Effect:
    $granted := true$

7.2 The Complete Implementation

In the rest of Section 7, we assume that $U = set_{\mathcal{C}}$, $K = [B2_{\mathcal{C}}]$ (the set of doubly-exponentiated bases), $X = [XConst1_{\mathcal{C}}] \cup [XConst2_{\mathcal{C}}]$, and $N = K \cup X$.

The implementation consists of two endpoint automata, an insecure channel, an eavesdropper and an environment. Specifically, implementation $S_2(\mathcal{C}, P, A)$ is constructed by composing the following automata:

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

and then hiding all the *eavesdrop*, *IC-send*, *IC-receive*, and *learn* actions. That is, we hide all but the external actions of $KD(U, P, K, A)$, which are the *grant* and *reveal* actions. Figure 2 contains an interaction diagram for S_2.

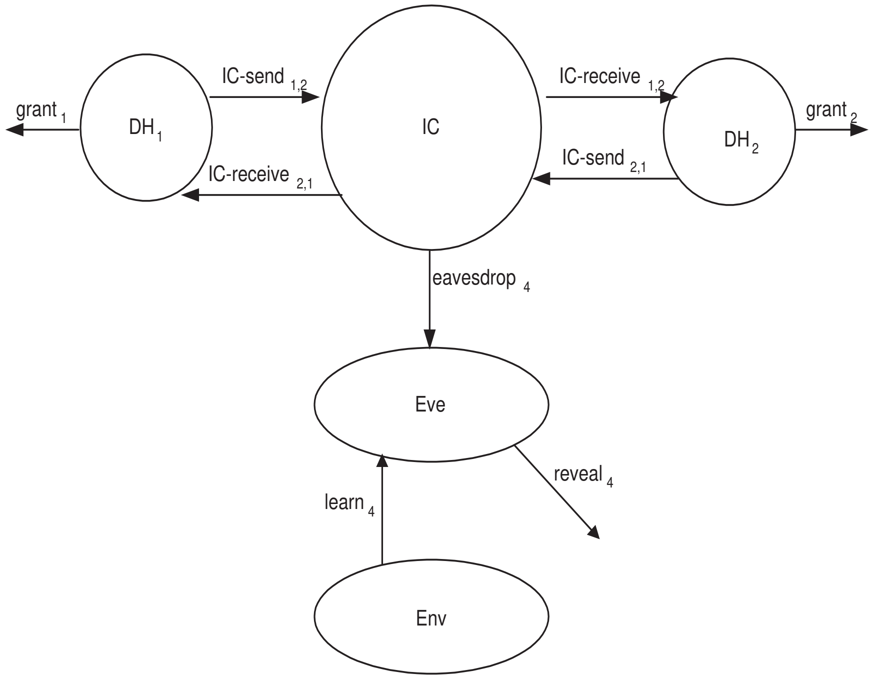

Figure 2: S_2; P ={1,2}; A ={4}

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 $DH(p).rcvd\text{-}base = DH(q).chosen\text{-}base$.*

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

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

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 $u \in N$, $u \notin IC.buffer(p,q)$.*

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

Proof: Analogous to the proof of Lemma 6.4. ∎

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. $t.\mathit{chosen\text{-}key} = \mathit{exp}(s.DH(p1).\mathit{chosen\text{-}base}, s.DH(p2).\mathit{chosen\text{-}exp})$ if $s.DH(p1).\mathit{chosen\text{-}exp} \neq \perp$ and $s.DH(p2).\mathit{chosen\text{-}exp} \neq \perp$; $t.\mathit{chosen\text{-}key} = \perp$ otherwise.

2. $t.\mathit{notified} = \{p \in P : s.DH(p).\mathit{granted}\}$.

Condition 1 says that the chosen key in KD is obtained by doubly-exponentiating $b0$ with both the chosen exponents in the Diffie-Hellman protocol. If it is not the case that both exponents have been chosen, then the chosen key is undefined.

Theorem 7.3 *F is a simulation relation.*

Proof: *Start condition:* Easy.
Step condition: Consider (s, π, s') and t as usual, and consider cases. The most interesting cases are:

1. $\pi = \mathit{choose\text{-}exp}_p$.

   If $s.DH(q).\mathit{chosen\text{-}exp} = \perp$, 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-key*, with a chosen value of
   $\mathit{exp}(s'.DH(p1).\mathit{chosen\text{-}base}, s'.DH(p2).\mathit{chosen\text{-}exp})$.
   Enabling is straightforward, as is the preservation of the simulation relation.

2. $\pi = \mathit{grant}(b)_p$

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

3. $\pi = \mathit{reveal}(u)_a$

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

$\square$

Theorem 7.4 *$S_2(\mathcal{C}, P, A)$ implements $KD(U, P, K, A)$.*

Proof: By Theorems 7.3 and 3.1. $\square$

8 Algorithms Using Structured-Key Cryptosystems

In this section, we modify the implementations of private communication and of key distribution, S_1 and S_2, so that they use a common structured-key cryptosystem, rather than separate shared-key and base-exponent cryptosystems. We show that the resulting systems are still correct. The proofs use simulation relations to the original systems.

Throughout this section, and for the rest of the paper, we fix $\mathcal{C}$ to be any augmented structured-key cryptosystem.

8.1 Private Communication

We show that moving from a shared-key cryptosystem to a structured-key cryptosystem does not disturb the correctness of the simple shared-key communication protocol. The key idea is that the new mechanisms added to the cryptosystem do not contribute any new ways of computing messages of the original shared-key cryptosystem.

8.1.1 Notation and assumptions

Starting from the fixed augmented structured-key cryptosystem $\mathcal{C}$, we derive a shared-key cryptosystem $\mathcal{C}'$, by defining $MConst_{\mathcal{C}'} = MConst_{C}$ 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 that P is a set with at least 2 elements, A is a nonempty finite set, $U = set_{\mathcal{C}}$, $M = [MConst_{\mathcal{C}}]$, $K = [B2_{\mathcal{C}}]$, and $X = [XConst1_{\mathcal{C}}] \cup [XConst2_{\mathcal{C}}]$.

We also define W to be 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 (possibly 0) of *enc* operations with second arguments in $set_{\mathcal{C}}(\text{"}B\text{"}) - K$. Informally speaking, w is obtained by "wrapping" some message of the derived shared-key cryptosystem in a series of encryptions based on keys *not* in $B2$. Finally, we assume that $N = W \cup K \cup X$, $U' = U = set_{\mathcal{C}}$, and A' is a nonempty finite set, disjoint from A.

The set W is used to describe the elements of type "M" that the eavesdropper is not allowed to learn. We have chosen this particular set W because it has a simple definition, because it includes all elements of type "M" that could help the eavesdropper to compute elements that are supposed to remain unknown (the *MConsts*), and because it excludes values produced by other protocols with which the given protocol is to be composed. Other choices of W besides ours are possible.

8.1.2 New implementation

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 *KConsts*. Also, the parameters have new meanings, as defined just above.

$Enc3(\mathcal{C}, P)_{p,q}$ **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:
$IC\text{-}send(u)_{p,q}$, $m \in set_{\mathcal{C}}$

States:

buffer, a multiset of elements of $[MConst_{\mathcal{C}}]$, initially empty
$shared\text{-}key \in set_{\mathcal{C}}(\text{“}B\text{”}) \cup \{\perp\}$, initially $\perp$

Transitions:

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

$grant(u)_p$
Effect:
if $u \in set_{\mathcal{C}}(\text{“}B\text{”})$ 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*

$Dec(\mathcal{C}, P)_{p,q}$, **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 $set_{\mathcal{C}}(\text{“}M\text{”})$
$shared\text{-}key \in set_{\mathcal{C}}(\text{“}B\text{”}) \cup \{\perp\}$, initially $\perp$

Transitions:

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

$grant(u)_q$
Effect:
if $u \in set_{\mathcal{C}}(\text{“}B\text{”})$ 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*

We define S_3 to be the system 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 constructed by composing:

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

and then hiding all the *eavesdrop*, *IC-send*, *IC-receive*, *grant*, and *learn* actions, and the $reveal_a$ actions for $a \in A'$. That is, we hide all actions except the external actions of $PC(U, P, M, A)$, which are the *PC-send* and *PC-receive* actions and the $reveal_a$ actions for $a \in A$. We want to show that $S_3(\mathcal{C}, P, A, U', A')$ implements $PC(U, P, M, A)$.

8.1.3 Invariants

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

1. *For all p, $Enc3_{p,q}.shared\text{-}key \in K \cup \{\perp\}$.*
2. *For all p, $Dec3_{p,q}.shared\text{-}key \in K \cup \{\perp\}$.*

Proof: By induction; the only interesting case is *grant*, but this is straightforward from the precondition of *grant* in *KD*. ∎

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

1. *For all p, q, if $u \in IC.buffer(p, q)$ then $u = enc(m, k)$, where $m \in M$ and $k \in K$.*
2. *For all p, q, all $x \in X$, $x \notin IC.buffer(p, q)$.*

Proof: Part 1 is proved by induction, using Lemma 8.1. The interesting case is $IC\text{-}send_{p,q}$; this follows because the precondition (in Enc_3) implies that the message sent is of the indicated form. Part 2 follows from Part 1. ∎

Lemma 8.3 *In all reachable states of S_3, the following are true:*

1. *No element of X is in $Eve.has$.*

Proof: By induction. The interesting cases are:

1. $eavesdrop(u)_{p,q,a}$

   Lemma 8.2 implies that $u = enc(m, k)$ for some $m \in M$ and $k \in K$. This is not in X, which shows that the invariant is preserved.

2. $learn(u)_a$

   The precondition (in *Env*) implies that $u \notin X$, so this cannot cause a violation of the claim.

3. $compute(u, f)_a$

   Since no element of X can be computed in cryptosystem $\mathcal{C}$, the claim is preserved.

∎

The following lemma, Lemma 8.4, has a different style from the other invariants we have stated so far. It does not say that *no element* of W may appear in *has*. Rather, it says that *if* such an element, w, appears in *has*, then any element m of $set_{\mathcal{C}'}(\text{"}M\text{"})$ that is easily reachable from w, for example, the "unwrapped" element of $set_{\mathcal{C}'}(\text{"}M\text{"})$ from which w is constructed, must also be in *has*.

Also note that we do not give any invariants here saying that K or M elements do not appear in $Eve.has$, as we did in Section 6.3. This is because (in the spirit of modularity) we prefer to avoid re-proving facts for S_3 that have already been proved for S_1.

Lemma 8.4 *In all reachable states of S_3, the following are true:*

1. *Assume that $(M \cup K) \cap Eve.has = \emptyset$. If $w \in W \cap Eve.has$ and $m \in set_{\mathcal{C}'}(\text{"}M\text{"})$ is easily reachable from $\{w\} \cup (set_{\mathcal{C}}(\text{"}B\text{"}) - K)$ in $\mathcal{C}$, then $m \in Eve.has$.*

Proof: By induction. Fix (s, π, s') as usual. The interesting cases are:

1. $eavesdrop(u)_{p,q,a}$

   Note that the form of u, as described in Lemma 8.2, implies that $u \in W$. The interesting situation is where w, the element in the statement of the invariant, is equal to u, the element newly inserted into *has*. So suppose that $m \in set_{\mathcal{C}'}$ is easily reachable from $\{u\} \cup (set_{\mathcal{C}'}(\text{"}B\text{"}) - K)$ in $\mathcal{C}$. Then properties of the cryptosystems $\mathcal{C}$ and $\mathcal{C}'$ imply that $m = u$. But u is explicitly put into $Eve.has$ by this step, as needed.

2. $learn(u)_a$

   The precondition (in *Env*) implies that $u \notin W$, so this cannot cause a violation of the claim.

3. $compute(u, f)_a$

   The interesting case is where the new element u being computed is in W, so assume that $u \in W$. Suppose that $(M \cup K) \cap s'.Eve.has = \emptyset$ and that $m \in set_{\mathcal{C}'}$ is easily reachable from $\{u\} \cup (set_{\mathcal{C}}(\text{"}B\text{"}) - K)$ in $\mathcal{C}$. It follows that $(M \cup K) \cap s.Eve.has = \emptyset$.

   If the function f is *exp*, then some element of X must be in $s.Eve.has$. But Lemma 8.3 implies that there is no such element. So f must be either *enc* or *dec*. Since no element of K is in $s.Eve.has$, the second argument in the application of f must be in $set_{\mathcal{C}}(\text{"}B\text{"}) - K$. The first argument is some $u' \in s.Eve.has$. It follows that m is easily reachable from $\{u'\} \cup (set_{\mathcal{C}}(\text{"}B\text{"}) - K)$ in $\mathcal{C}$. Also, since $u \in W$, properties of the cryptosystem $\mathcal{C}$ imply that also $u' \in W$. But then the inductive hypothesis implies that $m \in s.Eve.has$. Therefore, $m \in s'.Eve.has$, as needed.

$\square$

8.1.4 Implementation proof

We prove the correctness of S_3 as a consequence of that of $S_1(\mathcal{C}', P, A, U', A')$. By our previous result about S_1, Theorem 6.6:

Lemma 8.5 *$S_1(\mathcal{C}', P, A, U', A')$ 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 $reveal(u)_a$, $u \in U - set_{\mathcal{C}'}$, $a \in A$.

Lemma 8.6 *If β is a trace of $S_3(\mathcal{C}, P, A, U', A')$ then β 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, the states are identical in s and t.
2. $s.Eve.has \cap set_{\mathcal{C}'} \subseteq t.Eve.has$.

Theorem 8.7 *F is a simulation relation.*

Proof: *Start condition:* Easy.
Step condition: Consider (s, π, s') and t as usual.

We claim first that no element $u \in M \cup K$ appears in $s.Eve.has$. For, if such an element did appear in $s.Eve.has$, the fact that $(s,t) \in F$ would imply that $u \in t.Eve.has$. But this would contradict Lemma 6.4, an invariant for S_1.

The most interesting cases are:

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

   We consider two subcases:

   (a) $u \in set_{\mathcal{C}'}$
   Then the corresponding fragment consists of a single step, with the same action. The precondition of π in S_3' implies that $u \in s.Eve.has$. Since $(s,t) \in F$, we have also that $u \in t.Eve.has$. Therefore, π is enabled in t. Since *reveal* actions have no effect on the state, the relation F is preserved.

   (b) $u \in U - set_{\mathcal{C}'}$
   Then the corresponding fragment consists of the single state t. Since π is an internal action of S_3', the external behavior corresponds as needed.

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

   (a) $u \in set_{\mathcal{C}'}$
   Then since $f \in EN_{\mathcal{C}}$, f must be *exp*, *enc*, *dec*, or an element of *BConst*. We consider cases:

      i. $f = enc$ or $f = dec$, with the second argument in K.
      Then the precondition implies that this K element, say k, must be in $s.Eve.has$. But this contradicts a claim at the beginning of the proof, which means that this case cannot occur.

ii. $f = enc$, with the second argument in $set_{\mathcal{C}}(\text{“}B\text{”}) - K$
Then properties of the cryptosystem $\mathcal{C}$ imply that f yields a result $u \in (U - set_{\mathcal{C}'})$, contradicting the requirements of this case.

iii. $f = dec$, with the second argument in $set_{\mathcal{C}}(\text{“}B\text{”}) - K$
Then the corresponding fragment consists of the single state t. We must show that the correspondence is preserved. By properties of the cryptosystem $\mathcal{C}$, the first argument of f must be some element $w \in W$. By the precondition of π, $w \in s.Eve.has$. Then Lemma 8.4, together with the fact that $(M \cup K) \cap s.Eve.has = \emptyset$, implies that $u \in s.Eve.has$, that is, u is already in the *has* set, before the current step. Therefore, since $(s, t) \in F$, we have also $u \in t.Eve.has$. It follows that $(s', t) \in F$, that is, the correspondence is preserved.

iv. $f = exp$
Then f must be applied with a second argument $x \in X$, and $x \in s.Eve.has$. But this violates an invariant for S'_3, Lemma 8.3, which means that this case cannot occur.

v. $f \in BConst_{\mathcal{C}'}$
This yields a result $u \in (U - set_{\mathcal{C}'})$, contradicting the requirements of this case.

(b) $u \in U - set_{\mathcal{C}'}$
Then the corresponding fragment consists of the single state t. Since $u \notin set_{\mathcal{C}'}$, the correspondence is preserved.

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

(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. Since the same element is added to both *has* sets, the correspondence is preserved.

(b) $u \in U - set_{\mathcal{C}'}$
Then the corresponding fragment consists of the single state t. Since $u \notin set_{\mathcal{C}'}$, it is easy to see that the correspondence is preserved.

4. $\pi = eavesdrop(u)_{p,q,a}$
The precondition of π in S'_3 implies that $u \in s.IC.buffer_{p,q}$. Since $(s, t) \in F$, we have that also $u \in t.IC.buffer_{p,q}$. Therefore, π is enabled in t. Since the same element is added to both *has* sets, the correspondence is preserved.

$\square$

Theorem 8.8 *$S'_3(\mathcal{C}, P, A, U', A')$ implements $S_1(\mathcal{C}', P, A, U', A')$.*

Proof: By Theorems 8.7 and 3.1. $\square$

Lemma 8.9 *If β is a trace of $S_3(\mathcal{C}, P, A, U', A')$ then β 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 8.8 and Lemma 8.6. ∎

Theorem 8.10 *$S_3(\mathcal{C}, P, A, U', A')$ implements $PC(U, P, M, A)$.*

Proof: Let β be a trace of $S_3(\mathcal{C}, P, A, U', A')$. Then Lemma 8.9 implies that β_1 is a trace of $S_1(\mathcal{C}', P, A, U', A')$, where β_1 is equal to β with all *reveal*(u) actions removed, for $u \in U - set_{\mathcal{C}'}$. Then Lemma 8.5 implies that β_1 is a trace of $PC(set_{\mathcal{C}'}, P, M, A)$. It follows that β_1 is a trace of $PC(set_{\mathcal{C}}, P, M, A)$. Now, since β differs from β_1 only by including some *reveal* actions for elements in $U - set_{\mathcal{C}'}$, it follows that β 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

It is not hard to see that moving from a base-exponent cryptosystem to a structured-key cryptosystem does not disturb the correctness of the Diffie-Hellman protocol. The key idea is that the new mechanisms added to the cryptosystem involve the new message type "M", and do not contribute any new ways of computing bases or exponents.

We proceed formally as in the previous subsection. Starting from the fixed augmented structured-key cryptosystem $\mathcal{C}$, we derive an augmented base-exponent cryptosystem $\mathcal{C}'$ by defining $BConst_{\mathcal{C}'} = BConst_{\mathcal{C}}$, $XConst1_{\mathcal{C}'} = XConst1_{\mathcal{C}}$, $XConst2_{\mathcal{C}'} = XConst2_{\mathcal{C}}$, and $b0_{\mathcal{C}'} = b0_{\mathcal{C}}$. In this subsection we assume that $P = \{p1, p2\}$, A is a nonempty finite set, $U = set_{\mathcal{C}}$, $K = [B2_{\mathcal{C}}]$, $X = [XConst1_{\mathcal{C}}] \cup [XConst2_{\mathcal{C}}]$, and $N = K \cup X$.

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 system 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 constructed by composing:

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

and then hiding the *eavesdrop*, *IC-send*, *IC-receive*, and *learn* actions. That is, we hide all actions except the external actions of $KD(U, P, K, A)$, which are the *grant* and *reveal* actions. We want to show that $S_4(\mathcal{C}, P, A)$ implements $KD(U, P, K, A)$. We show this as a consequence of the correctness of $S_2(\mathcal{C}', P, A)$. By our previous result about S_2, Theorem 7.4:

Lemma 8.11 *$S_2(\mathcal{C}', P, A)$ implements $KD(set_{\mathcal{C}'}, P, K, A)$.*

In order to prove correctness of $S_4(\mathcal{C}, P, A)$, we would like to demonstrate a simulation relationship from $S_4(\mathcal{C}, P, A)$ to $S_2(\mathcal{C}', P, A)$. We define $S_4'(\mathcal{C}, P, A)$ from S_4 by hiding the actions $reveal(u)_a$, $u \in U - set_{\mathcal{C}'}$, $a \in A$.

Lemma 8.12 *If β is a trace of $S_4(\mathcal{C}, P, A)$ then β 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*, the states are identical in s and t.
2. $s.Eve.has \cap set_{\mathcal{C}'} \subseteq t.Eve.has$.

Theorem 8.13 *F is a simulation relation.*

Proof: Analogous to that of Theorem 8.7.
Start condition: Easy.
Step condition: Consider (s, π, s') and t as usual. The most interesting cases are:

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

   Analogous to the *reveal* case in the proof of Theorem 8.7.

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

   (a) $u \in set_{\mathcal{C}'}$
   Then properties of the structured-key cryptosystem imply that f must be either *exp* or an element of *BConst*. Moreover, any arguments required by f are also in $set_{\mathcal{C}'}$. Since such arguments must be in $s.Eve.has$ (by the enabling condition), the definition of F implies that they are also in $t.Eve.has$. It follows that π is enabled in t.
   Thus, we may allow the corresponding fragment to consist of a single step, with the same action. Since the same element is added to both *has* sets, the correspondence is preserved.

   (b) $u \in U - set_{\mathcal{C}'}$
   Analogous to the corresponding case for the *compute* action in the proof of Theorem 8.7.

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

   (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 $u \notin N = K \cup X$, by the precondition in S_4'. This implies that $learn(u)$ is enabled in S_2. Since the same element is added to both *has* sets, the correspondence is preserved.

   (b) $u \in U - set_{\mathcal{C}'}$
   Then the corresponding fragment consists of the single state t. Since $u \notin set_{\mathcal{C}'}$, it is easy to see that the correspondence is preserved.

4. $\pi = eavesdrop(u)_{p,q,a}$

   Analogous to the *eavesdrop* case in the proof of Theorem 8.7.

∎

Theorem 8.14 *$S_4'(\mathcal{C}, P, A)$ implements $S_2(\mathcal{C}', P, A)$.*

Proof: By Theorems 8.13 and 3.1. ∎

Lemma 8.15 *If β is a trace of $S_4(\mathcal{C}, P, A)$ then β with all $reveal(u)$ actions removed, for $u \in U - set_{\mathcal{C}'}$, is a trace of $S_2(\mathcal{C}', P, A)$.*

Proof: By Theorem 8.14 and Lemma 8.12. ∎

Theorem 8.16 *$S_4(\mathcal{C}, P, A)$ implements $KD(U, P, K, A)$.*

Proof: Let β be a trace of $S_4(\mathcal{C}, P, A)$. Then Lemma 8.15 implies that β_1 is a trace of $S_2(\mathcal{C}', P, A)$, where β_1 is equal to β with all *reveal*(u) actions removed, for $u \in U - set_{\mathcal{C}'}$. Then Lemma 8.11 implies that β_1 is a trace of $KD(set_{\mathcal{C}'}, P, K, A)$. It follows that β_1 is a trace of $KD(set_{\mathcal{C}}, P, K, A)$. Now, since β differs from β_1 only by including some *reveal* actions for elements in $U - set_{\mathcal{C}'}$, it follows that β is a trace of $KD(set_{\mathcal{C}}, P, K, A)$. ∎

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 using ordinary composition, 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.

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 assume, for the rest of the paper, that $U = set_{\mathcal{C}}$, $P = \{p1, p2\}$, $P' = \{p1', p2'\}$, A is a nonempty finite set, $M = [MConst_{\mathcal{C}}]$, $K = [B2_{\mathcal{C}}]$, $X = [XConst1_{\mathcal{C}}] \cup [XConst2_{\mathcal{C}}]$, W is the set of elements of $set_{\mathcal{C}}(\text{“}M\text{”})$ that can be obtained from elements of $set_{\mathcal{C}'}(\text{“}M\text{”}) \cup (set_{\mathcal{C}}(\text{“}B\text{”}) - K)$ in $\mathcal{C}$ using *enc*, $N = W \cup K \cup X$, and A' is a nonempty finite set, disjoint from A.

The combined system S_5 is constructed by composing:

- $Enc3(\mathcal{C}, P)_{p,q}$, $Dec3(\mathcal{C}, P)_{p,q}$, $p, q \in P$, $p \neq q$.
- $DH5_p$, $p \in P$; 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.

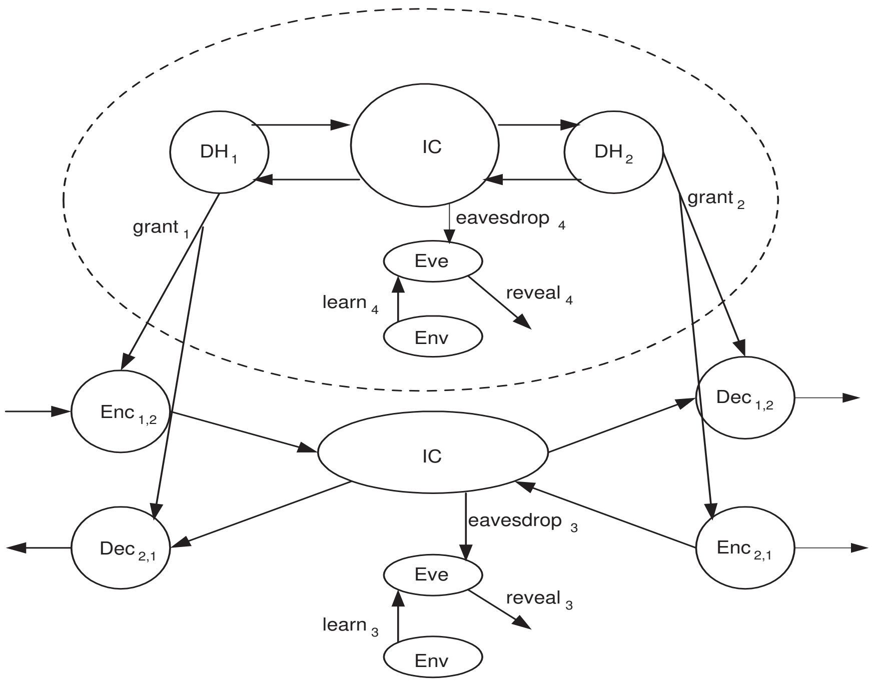

Figure 3: S_5

- $IC(U, P, A)$, $Eve(\mathcal{C}, P, A)$, $Env(U, A, N)$.
- $IC(U, P', A')$, $Eve(\mathcal{C}, P', A')$, $Env(U, A', N')$.

and hiding all actions except for the external actions of $PC(U, P, M, A)$, which are the *PC-send*, *PC-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 8.16 and 8.10, using general projection and pasting lemmas for I/O automata. Let α be an execution of S_5. We produce an execution α' of $PC(U, P, M, A)$ such that $trace(\alpha') = trace(\alpha)$.

Define $T_1 = Enc3_{p1,p2} \times Dec3_{p1,p2} \times Enc3_{p2,p1} \times Dec3_{p2,p1} \times IC(U, P, A) \times Eve(\mathcal{C}, P, A) \times Env(U, A, N)$ and $T_2 = DH5_{p1} \times DH5_{p2} \times IC(U, P', A') \times Eve(\mathcal{C}, P', A') \times Env(U, A', N')$. Define $\alpha_1 = \alpha | T_1$ and $\alpha_2 = \alpha | T_2$. By I/O automaton projection lemmas, α_1 and α_2 are executions of T_1 and T_2, respectively. (See [23], Chapter 8.)

Let T_3 be the same as T_2 but with all actions except for the *grant* actions and *reveal* actions hidden; α_2 is also an execution of T_3. Then T_3 is exactly the same as $S_4(\mathcal{C}, P, A')$ except for renaming of elements of P, everywhere except in *grant* actions, to corresponding elements of P'. By Theorem 8.16, $S_4(\mathcal{C}, P, A')$ implements $KD(U, P, K, A')$. Since all the renaming happens internally, this implies that T_3 implements $KD(U, P, K, A')$.

It follows that there exists an execution α_3 of $KD(U, P, K, A')$ that agrees with α_2, and so also with α, on the external actions of $KD(U, P, K, A')$, that is, on the $grant(u)_p$ actions, $u \in U$, $p \in P$ and $reveal(u)_a$ actions, $u \in U$, $a \in A'$.

Now I/O automaton pasting lemmas (see [23]) yield an execution α_4 of $T_1 \times KD(U, P, K, A')$ such that $\alpha_4 | T_1 = \alpha_1$ and $\alpha_4 | KD(U, P, K, A') = \alpha_3$. Thus, α_4 agrees with α on T_1 and on the external actions of $KD(U, P, K, A')$.

Now define $T_4 = T_1 \times KD(U, P, K, A')$, with all except the *PC-send*, *PC-receive* and *reveal*(a), $a \in A$ actions hidden. Note that $T_4 = S_3(\mathcal{C}, P, A, U, A')$. Now Theorem 8.10 implies that $S_3(\mathcal{C}, P, A, U, A')$ implements $PC(U, P, M, A)$; therefore, there is an execution α' of $PC(U, P, M, A)$ that agrees with α_4 on all external actions of $PC(U, P, M, A)$. Hence, α' agrees with α on all external actions of $PC(U, P, M, A)$. This is as needed. ∎

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. 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-send*, *PC-receive*, and $reveal_a$ actions for $a \in A$, that is, the external actions of $PC(U, P, M, A)$.

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). We define a simulation relation F from S_7 to S_5, where $(s, t) \in F$ exactly if:

1. For all except the *Eve* components, the states are identical in s and t.
2. $s.has \subseteq t.Eve(\mathcal{C}, P, A).has$.
3. $s.has \subseteq t.Eve(\mathcal{C}, P', A').has$.

This relation says, essentially, that any information that the combined eavesdropper can acquire, in the context of the given protocols, is something that each of the individual eavesdroppers could acquire anyway.

Theorem 9.2 *F is a simulation relation.*

Proof: The initial condition is immediate, because $s.has$ is empty. For the step condition, the interesting cases are as follows. Let b and b' be arbitrary elements of A and A', respectively.

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

   We know that $u \in s.has$. So by definition of F, we have that $u \in t.Eve(\mathcal{C}, P, A).has$. Let this step correspond to a single step, with the same $reveal(u)_a$ action. Since $u \in t.Eve(\mathcal{C}, P, A).has$, this action is enabled in S_5.

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

   Analogous to the previous case.

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

   The execution fragment corresponding to this step consists of two steps, with actions $learn(u)_b, learn(u)_{b'}$. By the precondition, $u \in U - (N \cup N')$. So $u \in (U - N)$ and $u \in (U - N')$. It follows that the two $learn$ actions are enabled in S_5. Since the same element u is added to all three has sets, the correspondence is preserved.

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

   The execution fragment corresponding to this step consists of two steps, with actions $compute(u, f)_b, compute(u, f)_{b'}$. The precondition implies that all the arguments needed for this computation of f are in $s.has$. Since $(s, t) \in F$, these areguments are also in $t.Eve(\mathcal{C}, P, A).has$ and in $t.Eve(\mathcal{C}, P', A').has$. It follows that the two $compute$ actions are enabled in S_5. Since the same element u is added to all three has sets, the correspondence is preserved.

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

   Then Lemma 8.2 implies that u is of the form $enc(m, k)$, $m \in M$, $k \in K$. Therefore, $u \in U - N'$. The corresponding fragment consists of two steps, with actions $eavesdrop(u)_a, learn(u)_{b'}$. The $eavesdrop$ action is enabled in S_5 because the channel states are identical in states s and t. The $learn$ action is enabled in S_7 because $u \in U - N'$. Again, since the same element u is added to all three has sets, the correspondence is preserved.

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

   Then u is of the form $exp(b0, x) \in U - N$. The corresponding fragment consists of two steps, with actions $eavesdrop(u)'_a, learn(u)_b$. The $eavesdrop$ action is enabled in S_5 because the channel states are identical in states s and t. The $learn$ action is enabled in S_7 because $u \in U - N$. Since the same element u is added to all three has sets, the correspondence is preserved.

∎

Theorem 9.3 *S_7 implements S_5.*

Proof: By Theorems 9.2 and 3.1. ∎

The essence of Theorems 9.2 and 9.3 is a relationship between $Eve(\mathcal{C}, P, A, P', A')$ and the composition $Eve(\mathcal{C}, P, A) \times Eve(\mathcal{C}, P', A')$. The reason that this relationship is described in terms of the complete systems S_7 and S_5 (rather than just the eavesdroppers) is that the relationship depends on assumptions about the contexts in which the eavesdroppers run. The key facts used about the contexts appear in the arguments for the *eavesdrop* cases in the proof of Theorem 9.2. Basically, these facts say that any message that can appear in the insecure channel of either protocol could also be generated by the environment in the other protocol. It is possible to extract a general combining theorem for eavesdroppers, using abstract models of the environments and protocols that express just this type of noninterference. Since this involves some notational complexities, we leave this for future work.

Lemma 9.4 *S_6 implements S_7.*

The fact that S_6 implements S_7 is easy, based on the following two lemmas:

Lemma 9.5 *$Env(U, A \cup A', N \cup N')$ implements $Env(U, A, N) \times Env(U, A', N')$.*

Lemma 9.6 *$IC(U, P, A, P', A')$ implements $IC(U, P, A) \times IC(U, P', A')$.*

This all yields:

Lemma 9.7 *S_6 implements S_5.*

Proof: By Lemma 9.4 and Theorem 9.3. ∎

Theorem 9.8 *S_6 implements $PC(U, P, M, A)$.*

Proof: By Lemmas 9.7 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. Even though this example is very simple, we have studied it using many kinds of decomposition, including:

1. Separating distributed algorithms issues from other issues, like cryptosystem reachability issues.

2. Treating the two sub-protocols separately, then combining them using general theorems about automaton composition.

3. Giving high level service specifications, giving detailed descriptions of implementing algorithms, and using simulation relations to show that the algorithms implement the services.

4. First studying the protocols using simple cryptosystems and later extending them to use more elaborate cryptosystems.

5. Combining separate adversaries into one.

We believe that understanding these decomposition methods in a simple context is an important first step toward extending them to more complicated protocols.

It appears possible to decompose the presentation in this paper even more. For example, one might define a notion of embeddings of cryptosystems and obtain the results of Section 8 as consequences of general theorems about such embeddings. Or, one might formulate and prove a general combining theorem for eavesdroppers and use it in the proof of Lemma 9.4.

In work in progress, we are extending these ideas to more complex protocols like that of Diffie, van Oorschot, and Weiner [11], 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 [11] are: The cryptosystems are more complicated, so more complicated arguments must 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 appropriate to combine the adversary and communication system 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 allowed. 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 addressed 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 [5, 22]. For more active adversaries, more sophisticated algorithms can 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. A great deal of reasoning about security protocols can be carried out in a framework in which it is assumed that certain low probability "bad" events simply do not occur. Such events might then be introduced separately, and general theorems used to limit their impact on system behavior. Such general theorems remain to be developed.

Acknowledgments: I thank Ron Rivest for getting me started on this project and for some very helpful discussions. Martin Abadi, Oleg Sheyner, Alessandro Gencarelli, Butler Lampson, Victor Luchangco, Anna Lysyanskaya, Dahlia Malkhi, Mike Reiter, Roberto Segala, and Jeannette Wing provided useful comments and encouragement.

References

[1] Martín 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] Martín Abadi, Michael Burrows, and Roger Needham. A logic of authentification. In *Proceedings of the Royal Society*, A,426,1871, pages 233–271, December 1989. Also appeared as SRC Research Report 39 and, in shortened form, in *ACM Transactions on Computer Systems*, 8, 1 (February 1990), 18-36.

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

[4] Martín Abadi and Mark R. Tuttle. A semantics for a logic of authentification. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, Montreal, Quebec, Canada, August 19-21 1991.

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

[6] M. Bellare and P. Rogaway. Entity authentification and key distribution. *Advances in Cryptology - CRYPTO'93*, 773, 1994.

[7] M. Bellare and P. Rogaway. Provably secure session key distribution - the three party case. In *Proceedings of the 27th ACM Symmposium on the Theory of Computing*, 1995.

[8] A. Cavalca. Tecnica dei vincoli negativi: un nuovo metodo per l'analisi di protocolli di autenticazione. Master's thesis, University of Bologna, October 1997.

[9] A. Cavalca and R. Segala. Negative constraints for the analysis of authentication protocols. Technical report, University of Bologna. To appear.

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

[11] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentification and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[12] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1993.

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

[14] Alan Fekete, Nancy Lynch, and Alex 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 [15].

[15] Alan Fekete, Nancy Lynch, and Alex 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.

[16] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing*, 18(1):186–208, February 1989.

[17] Jason Hickey, Nancy Lynch, and Robbert van Renesse. Specifications and proofs for Ensemble layers. In Rance 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.

[18] Mei Lin Hui and Gavin Lowe. Safe simplifying transformations for security protocols or not just the Needham Schroeder public key protocol. In *12th IEEE Computer Security Foundations Workshop (CSFW12)*, pages 32–43, Mordano, Italy, June 28-30 1999.

[19] Richard A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.

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

[21] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *5th ACM Conference on Computer and Communications Security*, pages 112–121, San Francisco, CA, USA, November 1998.

[22] Victor 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.

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

[24] Nancy Lynch. I/O automaton models and proofs for shared-key communication systems. In *12th IEEE Computer Security Foundations Workshop (CSFW12)*, pages 14–29, Mordano, Italy, June 28-30 1999.

[25] Nancy Lynch and Sergio 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.

[26] Nancy A. Lynch and Mark 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.

[27] Catherine Meadows. A system for the specification and analysis of key management protocols. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 182–195, 1991.

[28] Jonathan K. Millen, Sidney C. Clark, and Sheryl B. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, February 1987.

[29] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I. *Information and Computation*, 100(1):1–40, 1992.

[30] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[31] Lawrence C. Paulson. The Isabelle reference manual. Technical Report 283, University of Cambridge, Computer Laboratory, 1993.

[32] Tsvetomir P. Petrov, Anna Pogosyants, Stephen J. Garland, Victor Luchangco, and Nancy A. Lynch. Computer-assisted verification of an algorithm for concurrent timestamps. In Reinhard Gotzhein and Jan 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.

[33] Steve Schneider. Verifying authentication protocols with CSP. In *10th Computer Security Foundations Workshop*, pages 3–17. IEEE Computer Society Press, 1997.

[34] Oleg Sheyner and Jeannette Wing, 1999. Personal communication.

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

[36] Thomas Y. C. Woo and Simon S. Lam. A semantic model for authentication protocols. pages 178–194, 1993.