Secrecy by Typing in Security Protocols

Martín Abadi
ma@pa.dec.com

Digital Equipment Corporation
Systems Research Center

Abstract. We develop principles and rules for achieving secrecy properties in security protocols. Our approach is based on traditional classification techniques, and extends those techniques to handle concurrent processes that use shared-key cryptography. The rules have the form of typing rules for a basic concurrent language with cryptographic primitives, the spi calculus. They guarantee that, if a protocol typechecks, then it does not leak its secret inputs.

1 Introduction

Security is an elusive cocktail with many rare ingredients. For any given security protocol, one may want properties of integrity, confidentiality, availability, various forms of anonymity and non-repudiation, and more. Seldom does a protocol achieve all the properties that its designers intended. Moreover, even when a protocol is sound, it is often delicate to define all its important properties, and difficult to prove them.

The tasks of protocol design and analysis can be simplified by having principles that make it easier to achieve particular security objectives, in isolation, and rules that help recognize when these objectives have been achieved. For example, if we wish to obtain some availability properties and are concerned about denial of service attacks, we may design our protocols in such a way that there is an obvious, small bound on the amount of work that any principal will do in response to any message. One could relax this bound for messages from trusted, authenticated principals; but trust and authentication are not always correct, so letting availability depend on them should not be done lightly.

In this paper, we develop informal principles and formal rules for achieving secrecy properties in security protocols. These principles and rules are based on traditional concepts of classification and information flow [Den82,Gas88], extended to deal with concurrent processes that use shared-key cryptography. In particular, in analyzing a protocol, we label each piece of data and each communication channel as either secret or public. Secret data should not be sent on public channels, and secret channels should not be made available indiscriminately.

In our approach, encryption keys are pieces of data, and as such they are labelled. Secret data can be made public—declassified—by encryption under a secret key. However, this declassification cannot be done without some additional precautions. For example, given a secret bit b and a secret key K, we cannot simply publish b under K if we may also publish 0 or 1 under K: an attacker could deduce the value of b by comparing ciphertexts. The rules of this paper capture a sufficient set of simple precautions that permit this declassification.

The rules have the form of typing rules for a basic concurrent language, the spi calculus [AG97a]; this calculus is an extension of the pi calculus [MPW92] with shared-key cryptographic primitives. The purpose of these rules is rather different from those of standard typing rules for related languages (such as those of Pierce and Sangiorgi [PS96]). The rules guarantee that, if a protocol type-checks, then it does not leak its secret inputs. This secrecy is obtained independently of any other feature (or any flaw) of the protocol. For example, the rules may guarantee that a protocol does not leak the contents of certain messages, without concern for whether message replay is possible. The notion of leaking is formalized in terms of testing equivalence [DH84,BN95]: roughly, a process $P(x)$ does not leak the input x if a second process Q cannot distinguish running in parallel with $P(M)$ from running in parallel with $P(M')$, for every M and M'.

These typing rules are helpful, in that following them may lead to designs that are clearer, more robust, and more likely to be sound. Furthermore, they enable us to give simple proofs of properties that would be hard to establish from first principles. However, the secrecy theorems that we obtain should be taken with a grain of salt: they are dependent on the choice of a particular model, that of the spi calculus. This model is fairly accurate and expressive, but does not take into account issues of key length, for example.

The next section explains, informally, our approach for achieving secrecy. Section 3 is a review of the spi calculus. (The spi calculus presented is a generalization of that defined originally [AG97a]; it includes polyadic constructs [Mil91].) Section 4 provides the typing rules for the spi calculus, and Section 5 shows that these rules can be applied to prevent undesirable flows of information. Section 6 illustrates the use of the rules in examples. Finally, Section 7 discusses some conclusions.

Throughout this paper, we often opt for simplicity over generality, considering this work only a first exploration of a promising approach. In particular, we take a binary, Manichaean view of secrecy. According to this view, the world is divided into system and attacker, and a secret is something that the attacker does not have. A more sophisticated view of secrecy would distinguish various principals within the system, and enable us to discuss which of these principals have a given piece of data. In our binary view, however, we can vary the boundary between system and attacker, with some of the same benefits.

2 Some Principles for Secrecy

In the security literature there are well-known methods for controlling flows of information. Typically, these methods rely on putting objects and subjects into security classes, and guaranteeing that no data flows from higher classes to lower classes. In some of these methods, security classes are formalized as types, and the control of flows of information relies on typing rules (e.g., [VSI96]).

We adapt some of that work to security protocols. This section describes our approach informally, along with the main difficulties that it addresses. It is probable that some of the basic observations of this section have already been made, but they do not seem to be documented in the open literature.

2.1 On Rules in Distributed Systems

In a centralized system, an administrator that controls all the hardware in the system may hope to control all information flows. If all communication is mediated by the system hardware, the control of all information flows is plausible at least in principle. In particular, the administrator may check user programs before running them, statically, or may apply some dynamic checks.

In a distributed system, on the other hand, no single administration may control all the hardware. No part of the system may be able to check that the software of any other part is constructed according to a given set of rules. At best, each principal can analyze the programs that it receives from other principals, as well as all other messages.

Therefore, whatever rules we propose, they should be such that an attacker satisfies them vacuously. We cannot expect to restrict the code that the attacker runs. Our rules should constrain only the principals that want to protect their secrets from the attacker.

2.2 Preliminaries on Keys and Channels

As mentioned in the introduction, this paper concerns shared-key protocols. We write $\{M\}_K$ for the result of encrypting M with K, using a shared-key cryptosystem such as DES [DES77]. With shared-key cryptography, secrecy can be achieved by communication on public channels under secret keys.

In addition to public channels, on which anyone may communicate, we consider channels with some built-in protection. We restrict attention to channels on which the same principals can send and receive. On a single machine, channel protection can be provided by an operating system that mediates communication between user processes; in a distributed system, the protection can be implemented cryptographically.

The ability to communicate on a channel is often determined by possession of a capability, such as a password or a key. (In the pi calculus and the spi calculus, the name for a channel is the capability for the channel.) A public channel is one for which the capability is public; similarly, a secret channel is one for which the capability is secret.

2.3 Classifying Data

We consider only three classes of data:

- *Public* data, which can be communicated to anyone,
- *Secret* data, which should not be leaked,
- *Any* data, that is, arbitrary data.

We use the symbols T and R to range over the classes *Secret*, *Public*, and *Any*. We refer to *Secret*, *Public*, and *Any* as classes, levels, or types; the difference is one of emphasis at most. It should be possible to generalize our rules to richer classification structures, with more than three classes; but such a generalization is not essential for our immediate purposes.

Encryption keys are data, so our classification scheme applies to them. Several different levels can be associated with a key:

- its level as a piece of data,
- the levels of the data that it is used to encrypt,
- the levels of the resulting ciphertexts.

However, not all combinations are possible, as for example a public key should not be used to turn secret data into public ciphertexts. It is simplest to retain only one level for each key. Similarly, we associate a single classification with each channel. We adopt the following principles:

> The result of encrypting data with a public key has the same classification as the data, while the result of encrypting data with a secret key may be made public.
> Only public data can be sent on public channels, while all kinds of data may be sent on secret channels.

The relation $T <: R$ holds if T equals R or if R is *Any*. If a piece of data has level T and $T <: R$, then the piece of data has level R as well.

Because a piece of data of level *Any* could be of level *Secret*, it should not be leaked. On the other hand, a piece of data of level *Any* could be of level *Public*, so it cannot be used as a secret. For example, it cannot be used as a secret key for encrypting secret data. Thus, if all we know about a piece of data is that it has level *Any*, then we should protect it as if it had level *Secret*, but we can exploit it only as if it had level *Public*. This piece of data is therefore not very useful for constructing channel capabilities or encryption keys; we find it cleaner to forbid these uses altogether.

Furthermore, we focus on the case where the classification *Secret* is given only to data created fresh within the system that we are trying to protect. However, other data can be of class *Any*, and then it must be protected as though it were of class *Secret*.

2.4 Classifying Inputs

Whereas each principal may know the classification of the data that it creates, the principal may not always know the classification of the data that it acquires through communication with other principals. Data that arrives in clear on public channels can be presumed to be public. On the other hand, data that arrives with some protection may be either secret or public.

The participants of protocols typically know how to handle each of the fields of the encrypted messages that they receive, as in the following example (inspired by the Needham-Schroeder protocol [NS78]):

$$\begin{array}{lll} \text{Message 1} & A \to S: & A, B \\ \text{Message 2} & S \to A: & \{A, B, I_A, \{I_B\}_{K_{SB}}\}_{K_{SA}} \\ \text{Message 3} & A \to B: & \{I_B\}_{K_{SB}} \end{array}$$

In this protocol, the principals A and B share keys K_{SA} and K_{SB} with the server S, respectively. The server provides some confidential information I_A to A and I_B to B, in response to a request from A. When A receives a message from the server, it decrypts it, retrieves I_A, and forwards $\{I_B\}_{K_{SB}}$ to B. It is crucial that A be able to recognize when this action is appropriate. For example, if a principal X plays the role of A and B in two concurrent runs of the protocol, it should be able to recognize whether a message is an instance of Message 2 or an instance of Message 3. If X mistakes an instance $\{I_X\}_{K_{SX}}$ of Message 3 for an instance Message 2, then X would forward part of I_X in clear.

As in this example, it is common for principals to deduce the sensitivity of inputs from the expected kind of a message or other implicit information. Such implicit information is often incorrect and often hard to analyze [AN96].

It is clearer to label explicitly each component of a message with a classification, avoiding the dependence on implicit context. This labelling is important only for messages on secret channels or under secret keys, as all other messages can contain only public information.

Alternatively, we may adopt a standard format for all messages on secret channels or under secret keys. The format should guarantee that there is a standard way to attach classifications to parts of each message, avoiding again the dependence on implicit context. In our rules, we adopt this scheme. Each message on a secret channel has three components, the first of which has level *Secret*, the second *Any*, and the third *Public*. Each message under a secret key has those three components plus a confounder component, as discussed next.

Both of these schemes are implementations of the following principle:

> Upon receipt of a message, it should be easy to decide which parts of the contents are sensitive information, if any. This decision is least error-prone when it does not depend on implicit context.

2.5 Confounders

As noted in the introduction, given a secret bit b and a secret key K, we cannot simply publish b under K if we may also publish 0 or 1 under K: an attacker could deduce the value of b by comparing ciphertexts. On the other hand, we can create a fresh value n, then publish the concatenation of b and n under K. A value such as n is sometimes called a *confounder*. The purpose of n is to guarantee that the resulting ciphertext differs from all previous ones. The resulting ciphertext will differ from all future ones too if each ciphertext includes a fresh confounder, always in the same position. In general (unless one is concerned about encrypting known plaintext), confounders need not be kept secret.

Confounders are not needed if encryption is history-dependent, so a different transformation is applied to each message. In particular, confounders are not needed when encryption is done with a stream cipher or with a block cipher in an appropriate chaining mode. The remarks of this section are not intended to apply to protocols built on such algorithms.

Independently of the choice of cipher, confounders are not needed in protocols where all ciphertexts generated are guaranteed to be different. Unfortunately, it is not always clear whether this guarantee is offered.

As an example, we consider a simple protocol where a principal A sends a message M to a principal B under a shared key K_{AB}. Before A sends the message, B provides a challenge N_B, fresh for each message; the challenge serves as a proof of freshness, and protects against replays.

$$\begin{array}{ll} \text{Message 1} & B \rightarrow A: \ N_B \\ \text{Message 2} & A \rightarrow B: \ \{M, N_B\}_{K_{AB}} \end{array}$$

We may reason that all the ciphertexts from A are different, since each of them includes a fresh challenge. However, this reasoning is incorrect. An attacker C could provide a challenge instead of B, and A would reply without noticing that the challenge did not come from B. The attacker may pick the same challenge N_C twice in a row. In that case, the ciphertexts with which A responds, $\{M, N_C\}_{K_{AB}}$ and $\{M', N_C\}_{K_{AB}}$, are identical if and only if the cleartexts M and M' are identical. Thus, C can find out whether A is sending two identical cleartexts, even without knowing the key K_{AB}. Thus, the protocol is leaking some information.

In order to prevent this small leak, A should create a confounder N_A for each encryption. The modified protocol is:

$$\begin{array}{ll} \text{Message 1} & B \rightarrow A: \ N_B \\ \text{Message 2} & A \rightarrow B: \ \{M, N_B, N_A\}_{K_{AB}} \end{array}$$

This protocol is analyzed in [AG97c], where it is proved that the protocol guarantees the secrecy of M.

It is prudent to adopt the following principle:

> If each encrypted message of a protocol includes a freshly generated confounder in a standard position, then the protocol will not generate the same ciphertext more than once. Confounders should be used unless it is obvious that they are not needed.

2.6 Implicit Flows

A system may reveal something about one of its parameters even if this parameter never appears explicitly in a message. For example, the system may send different cleartext messages depending on the value of the parameter. An attacker could deduce something about the parameter from patterns of communication. Such a leak of information is sometimes called an *implicit flow*.

It is of course important to restrict implicit flows. For this purpose, we may forbid any comparison that involves pieces of sensitive data and that could be followed by actions that would reveal the result of the comparison.

It is also important not to push this restriction to uncomfortable extremes. Many protocols exhibit implicit flows in some form, usually without severe undesirable effects. As an example, we consider again this protocol:

$$\begin{array}{lll} \text{Message 1} & A \to S: & A, B \\ \text{Message 2} & S \to A: & \{A, B, I_A, \{I_B\}_{K_{SB}}\}_{K_{SA}} \\ \text{Message 3} & A \to B: & \{I_B\}_{K_{SB}} \end{array}$$

An attacker can send an arbitrary message to A, instead of Message 2. On receipt of this message, A performs a test that involves the secret K_{SA}, and branches on the result of this test. The reaction of A depends visibly on whether the message is a ciphertext under K_{SA}. One could regard this as an implicit flow, but perhaps one of little importance because the chance that an independently created message will be under K_{SA} is negligible. We can allow this implicit flow; it is harmless in our model.

Our (tentative) policy on implicit flows is summarized in the following principle:

> Implicit flows of information should be prevented, except perhaps when the likelihood of the implicit flow is no greater than the likelihood that an attacker will guess the information.

2.7 Further Principles

The discussion of this section is not comprehensive. In practice, several additional warnings and techniques are important. A few of these are:

- It is often hard not to leak the size of a secret. The use of padding in encrypted messages can help in this respect.
- It is prudent to minimize the benefit that an attacker may derive from discovering any one secret. In particular, the same key should not be used to protect a great volume of sensitive traffic, because then even a brute-force attack on the key may be profitable.
- Weak secrets, such as passwords, should be protected from brute-force attacks (see for example [Sch96]).

Undoubtedly there are more. However, the discussion of this section suffices as background for our rules and theorems.

3 The Untyped, Polyadic Spi Calculus

This section defines the version of the spi calculus that serves as the setting for our formal work. The main novelty with respect to the earlier versions of the spi calculus is the introduction of polyadic forms: each input, output, encryption, and decryption operation applies to an arbitrary number of pieces of data, rather than to a single piece. This novelty is important for the typing rules of the next section. However, this novelty is not substantial, as in particular it does not affect the expressiveness of the spi calculus.

Therefore, our presentation is mostly a review. Most of this material is derived from the earlier presentations of the spi calculus [AG97b]; it includes ideas common in the pi-calculus literature.

3.1 Syntax

We assume an infinite set of *names* and an infinite set of *variables*. We let m, n, p, q, and r range over names, and let w, x, y, and z range over variables. We write $P[M/x]$ for the outcome of replacing each free occurrence of x in P with M, and identify expressions up to renaming of bound variables and names.

The set of *terms* is defined by the grammar:

$L, M, N ::=$	terms
n	name
(M, N)	pair
0	zero
$suc(M)$	successor
$\{M_1, \ldots, M_k\}_N$	shared-key encryption ($k \geq 0$)
x	variable

The set of *processes* is defined by the grammar:

$P, Q, R ::=$	processes
$\overline{M}\langle N_1, \ldots, N_k\rangle.P$	output ($k \geq 0$)
$M(x_1, \ldots, x_k).P$	input ($k \geq 0$)
$P \mid Q$	parallel composition
$(\nu n)P$	restriction
$!P$	replication
$[M\ is\ N]\ P$	match
$\mathbf{0}$	nil
$let\ (x, y) = M\ in\ P$	pair splitting
$case\ M\ of\ 0 : P\ suc(x) : Q$	integer case
$case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P$	shared-key decryption ($k \geq 0$)

(The name n and the variables x, y, x_1, ..., x_k are bound in these processes.)

Most of these constructs should be familiar from earlier process algebras; see [AG97a] for a review, and see below for an operational semantics. The informal semantics of some of the constructs is as follows.

- The term $\{M_1, \ldots, M_k\}_N$ represents the ciphertext obtained by encrypting $M_1, \ldots, M_k$ under the key N.
- The process $case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P$ attempts to decrypt L with the key N; if L has the form $\{M_1, \ldots, M_k\}_N$, then the process behaves as $P[M_1/x_1] \ldots [M_k/x_k]$, and otherwise the process is stuck.
- The process $let\ (x, y) = M\ in\ P$ behaves as $P[N/x][L/y]$ if M is a pair (N, L), and it is stuck if M is not a pair.
- The process $case\ M\ of\ 0 : P\ suc(x) : Q$ behaves as P if M is 0, as $Q[N/x]$ if M is $suc(N)$ for some N, and otherwise is stuck.
- The process $[M\ is\ N]\ P$ behaves as P if M and N are equal, and otherwise is stuck.
- The process $\overline{M}\langle N_1, \ldots, N_k\rangle.P$ outputs $N_1, \ldots, N_k$ on M and then behaves as P; the output happens only if M is a name and there is another process ready to receive k inputs on M. We use $\overline{M}\langle N_1, \ldots, N_k\rangle$ as an abbreviation for $\overline{M}\langle N_1, \ldots, N_k\rangle.\mathbf{0}$, so $\overline{M}\langle N_1, \ldots, N_k\rangle$ is a process that may output $N_1, \ldots, N_k$ on M and then stop.
- The process $M(x_1, \ldots, x_k).P$ is ready to receive k inputs $N_1, \ldots, N_k$ on M, and then to behave as $P[N_1/x_1] \ldots [N_k/x_k]$.
- The process $!P$ behaves as infinitely many copies of P running in parallel.
- The process $(\nu n)P$ makes a new name n and then behaves as P.

The polyadic notations (input, output, encryption, decryption for $k \neq 1$) are not necessary for expressiveness. They are definable from the corresponding unary notations; for example, we can set:

$$\begin{aligned} c(x_1, x_2).P &\triangleq c(y).let\ (x_1, x_2) = y\ in\ P \\ case\ L\ of\ \{x_1, x_2\}_N\ in\ P &\triangleq case\ L\ of\ \{y\}_N\ in \\ &\qquad let\ (x_1, x_2) = y\ in\ P \end{aligned}$$

where variable y is fresh. However, as mentioned above, the polyadic constructs are useful for typing. They also introduce a typing difficulty, when arities do not match, as in $case\ \{M_1, M_2, M_3\}_N\ of\ \{x_1, x_2\}_N\ in\ P$. This typing difficulty could be overcome with an appropriate system of sorts.

In addition to the polyadic notations, we use the following standard abbreviations for any $k \geq 2$:

$$\begin{aligned} (N_1, \ldots, N_k, N_{k+1}) &\triangleq ((N_1, \ldots, N_k), N_{k+1}) \\ let\ (x_1, \ldots, x_k, x_{k+1}) = N\ in\ P &\triangleq let\ (y, x_{k+1}) = N\ in \\ &\qquad let\ (x_1, \ldots, x_k) = y\ in\ P \end{aligned}$$

where variable y is fresh.

We write $fn(M)$ and $fn(P)$ for the sets of names free in term M and process P respectively, and write $fv(M)$ and $fv(P)$ for the sets of variables free in M and P respectively. A term or process is closed if it has no free variables.

3.2 Commitment

As an operational semantics, we rely on a commitment relation. The definition of commitment depends on some new syntactic forms, *abstractions* and *concretions*. An abstraction is an expression of the form $(x_1, \ldots, x_k)P$, where $k \geq 0$, $x_1, \ldots, x_k$ are bound variables, and P is a process. A concretion is an expression of the form $(\nu m_1, \ldots, m_l)\langle M_1, \ldots, M_k\rangle P$, where $l, k \geq 0$, $M_1, \ldots, M_k$ are terms, P is a process, and the names $m_1, \ldots, m_l$ are bound in $M_1, \ldots, M_k$ and P. An *agent* is an abstraction, a process, or a concretion. We use the metavariables A and B to stand for arbitrary agents.

We extend the restriction and composition operators to abstractions:

$$\begin{aligned} (\nu m)(x_1, \ldots, x_k)P &\triangleq (x_1, \ldots, x_k)(\nu m)P \\ R \mid (x_1, \ldots, x_k)P &\triangleq (x_1, \ldots, x_k)(R \mid P) \end{aligned}$$

assuming that $x_1, \ldots, x_k \notin fv(R)$, and to concretions:

$$\begin{aligned} (\nu m)(\nu \vec{n})\langle \vec{M}\rangle Q &\triangleq (\nu m, \vec{n})\langle \vec{M}\rangle Q \\ R \mid (\nu \vec{n})\langle \vec{M}\rangle Q &\triangleq (\nu \vec{n})\langle \vec{M}\rangle (R \mid Q) \end{aligned}$$

assuming that $m \notin \{\vec{n}\}$ and $\{\vec{n}\} \cap fn(R) = \emptyset$. We define the dual composition $A \mid R$ symmetrically. (The definition of $(\nu m)(\nu \vec{n})\langle \vec{M}\rangle Q$ is slightly different from the original one [AG97b]. The change is crucial for Lemma 9, below.)

If F is the abstraction $(x_1, \ldots, x_k)P$ and C is the concretion $(\nu n_1, \ldots, n_l)\langle M_1, \ldots, M_k\rangle Q$, and $\{n_1, \ldots, n_l\} \cap fn(P) = \emptyset$, we define the processes $F@C$ and $C@F$ as follows:

$$\begin{aligned} F@C &\triangleq (\nu n_1) \ldots (\nu n_l)(P[M_1/x_1] \ldots [M_k/x_k] \mid Q) \\ C@F &\triangleq (\nu n_1) \ldots (\nu n_l)(Q \mid P[M_1/x_1] \ldots [M_k/x_k]) \end{aligned}$$

The *reduction relation* $>$ is the least relation on closed processes that satisfies the following axioms:

(Red Repl)
$$!P > P \mid !P$$
(Red Match)
$$[M \textit{ is } M]\, P > P$$
(Red Let)
$$\textit{let } (x, y) = (M, N) \textit{ in } P > P[M/x][N/y]$$
(Red Zero)
$$\textit{case } 0 \textit{ of } 0 : P \textit{ suc}(x) : Q > P$$
(Red Suc)
$$\textit{case suc}(M) \textit{ of } 0 : P \textit{ suc}(x) : Q > Q[M/x]$$
(Red Decrypt)
$$\textit{case } \{M_1, \ldots, M_k\}_N \textit{ of } \{x_1, \ldots, x_k\}_N \textit{ in } P > P[M_1/x_1] \ldots [M_k/x_k]$$

A *barb* is a name m (representing input) or a co-name $\overline{m}$ (representing output). An *action* is a barb or the distinguished *silent action* τ. The *commitment relation* is written $P \xrightarrow{\alpha} A$, where P is a closed process, α is an action, and A is a closed agent. It is defined by the rules:

(Comm Out)
$$\frac{}{\overline{m}\langle M_1, \ldots, M_k\rangle.P \xrightarrow{\overline{m}} (\nu)\langle M_1, \ldots, M_k\rangle P}$$

(Comm In)
$$\frac{}{m(x_1, \ldots, x_k).P \xrightarrow{m} (x_1, \ldots, x_k)P}$$

(Comm Inter 1)
$$\frac{P \xrightarrow{m} F \quad Q \xrightarrow{\overline{m}} C}{P \mid Q \xrightarrow{\tau} F@C}$$

(Comm Inter 2)
$$\frac{P \xrightarrow{\overline{m}} C \quad Q \xrightarrow{m} F}{P \mid Q \xrightarrow{\tau} C@F}$$

(Comm Par 1)
$$\frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q}$$

(Comm Par 2)
$$\frac{Q \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} P \mid A}$$

(Comm Res)
$$\frac{P \xrightarrow{\alpha} A \quad \alpha \notin \{m, \overline{m}\}}{(\nu m)P \xrightarrow{\alpha} (\nu m)A}$$

(Comm Red)
$$\frac{P > Q \quad Q \xrightarrow{\alpha} A}{P \xrightarrow{\alpha} A}$$

3.3 Testing Equivalence

A *test* is a pair (R, β) consisting of a closed process R and a barb β. We say that P *passes* a test (R, β) if and only if

$$(P \mid R) \xrightarrow{\tau} Q_1 \ldots \xrightarrow{\tau} Q_n \xrightarrow{\beta} A$$

for some $n \geq 0$, some processes $Q_1, \ldots, Q_n$, and some agent A. We obtain a testing preorder $\sqsubseteq$ and a testing equivalence $\simeq$ on closed processes:

$$\begin{aligned} P \sqsubseteq Q &\triangleq \text{ for any test } (R,\beta), \text{ if } P \text{ passes } (R,\beta) \text{ then } Q \text{ passes } (R,\beta) \\ P \simeq Q &\triangleq P \sqsubseteq Q \text{ and } Q \sqsubseteq P \end{aligned}$$

A *strict barbed simulation* is a binary relation $\mathcal{S}$ on closed processes such that $P \,\mathcal{S}\, Q$ implies:

(1) for every barb β, if $P \xrightarrow{\beta} A$ then $Q \xrightarrow{\beta} B$ for some B,
(2) if $P \xrightarrow{\tau} P'$ then there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \,\mathcal{S}\, Q'$.

(These requirements are somewhat stronger than those for barbed simulations [AG97b].) A *strict barbed bisimulation* is a relation $\mathcal{S}$ such that both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are strict barbed simulations.

The following lemma provides a method for proving testing equivalence:

Lemma 1. *If for every closed process R there exists a strict barbed bisimulation $\mathcal{S}$ such that $(P \mid R)\, \mathcal{S}\, (Q \mid R)$, then $P \simeq Q$.*

Proof. The lemma is a consequence of earlier results [AG97b], but we give a simple direct proof, since there is one. We argue that if P passes a test (R, β) then so does Q, assuming that there exists a strict barbed bisimulation $\mathcal{S}$ such that $(P \mid R)\, \mathcal{S}\, (Q \mid R)$. (Symmetrically, if Q passes a test (R, β) then so does P.)

If P passes (R, β), then

$$(P \mid R) \xrightarrow{\tau} R_1 \ldots \xrightarrow{\tau} R_n \xrightarrow{\beta} A$$

for some agent A and processes $R_1, \ldots, R_n$, for some $n \geq 0$. By the definition of strict barbed simulation, there exist $R'_1, \ldots, R'_n$ and A' such that $R_i\, \mathcal{S}\, R'_i$ for all $i \leq n$, and

$$(Q \mid R) \xrightarrow{\tau} R'_1 \ldots \xrightarrow{\tau} R'_n \xrightarrow{\beta} A'$$

Therefore, Q passes (R, β). $\square$

4 The Typing System

This section describes our rules for controlling information flow in the spi calculus. They are based on the ideas of Section 2. While there are several ways of formalizing those ideas, we embody them in a typing system for the spi calculus. This typing system is simple enough that it could be enforced statically. Our main results for this typing system are in Section 5.

The typing system gives rules for three kinds of assertions (or judgments):

- "$\vdash E$ well-formed" means that environment E is well-formed.
- "$E \vdash M : T$" means that term M is of class T in E.
- "$E \vdash P : Ok$" means that process P typechecks in E.

4.1 Environments

An environment is a list of distinct names and variables with associated levels. In addition, each name n has an associated term, of the form $\{M_1, \ldots, M_k, n\}_N$ for some $k \geq 0$. Intuitively, this association means that the name n may be used as a confounder only in the term $\{M_1, \ldots, M_k, n\}_N$. We write $x : T$ for variable x with level T, and $n : T :: \{M_1, \ldots, M_k, n\}_N$ for name n with level T and term $\{M_1, \ldots, M_k, n\}_N$. We write $n : T$ as a shorthand for $n : T :: \{n\}_0$, arbitrarily, when n is not needed as a confounder.

The set of names and variables declared in an environment E are its *domain*; we write it $dom(E)$.

The rules for environments are:

(Environment Empty)
$$\frac{}{\vdash \emptyset \text{ well-formed}}$$

(Environment Variable)
$$\frac{\vdash E \text{ well-formed} \qquad x \notin dom(E)}{\vdash E, x : T \text{ well-formed}}$$

(Environment Name)
$$\frac{\begin{array}{c}\vdash E \text{ well-formed} \qquad n \notin dom(E) \\ E \vdash M_1 : T_1 \ \ldots \ E \vdash M_k : T_k \qquad E \vdash N : R\end{array}}{\vdash E, n : T :: \{M_1, \ldots, M_k, n\}_N \text{ well-formed}}$$

Collectively, these rules enable us to form environments as lists of variables and names with associated levels, and in addition with terms attached to the names. The empty list is written $\emptyset$ in the rule (Environment Empty).

In the rule (Environment Name), the hypotheses $E \vdash M_1 : T_1, \ldots, E \vdash M_k : T_k$ are included so that, if any variable occurs in $M_1, \ldots, M_k$, then it is declared in E. This restriction is important because, without proper care, a variable could be instantiated in two different ways, so that the same confounder could be used for two different messages; this double use would defeat the purpose of confounders.

4.2 Terms

The rules for terms are:

(Level Subsumption)
$$\frac{E \vdash M : T \qquad T <: R}{E \vdash M : R}$$

(Level Variable)
$$\frac{\vdash E \text{ well-formed} \qquad x : T \text{ in } E}{E \vdash x : T}$$

(Level Name)
$$\frac{\vdash E \text{ well-formed} \qquad n : T :: \{M_1, \ldots, M_k, n\}_N \text{ in } E}{E \vdash n : T}$$

(Level Zero)
$$\frac{\vdash E \text{ well-formed}}{E \vdash 0 : \mathit{Public}}$$

(Level Successor)
$$\frac{E \vdash M : T}{E \vdash \mathit{suc}(M) : T}$$

(Level Pair)
$$\frac{E \vdash M : T \qquad E \vdash N : T}{E \vdash (M, N) : T}$$

(Level Encryption *Public*)
$$\frac{E \vdash M_1 : T \ \ldots \ E \vdash M_k : T \qquad E \vdash N : \mathit{Public}}{E \vdash \{M_1, \ldots, M_k\}_N : T}$$

(Level Encryption *Secret*)

$$\frac{E \vdash M_1 : \mathit{Secret} \qquad E \vdash M_2 : \mathit{Any} \qquad E \vdash M_3 : \mathit{Public} \qquad E \vdash N : \mathit{Secret} \qquad n : T :: \{M_1, M_2, M_3, n\}_N \text{ in } E}{E \vdash \{M_1, M_2, M_3, n\}_N : \mathit{Public}}$$

The rule (Level Subsumption) says that a term of level *Public* or *Secret* has level *Any* as well.

The rules (Level Variable) and (Level Name) enable us to extract the levels of names and variables from an environment.

The rules (Level Zero) says that 0 is of level *Public*. The rule (Level Successor) says that adding one preserves the level of a piece of data. Therefore, the terms 0, $suc(0)$, $suc(suc(0))$, ... are all of level *Public*. However, a term of the form $suc(x)$ may be of level *Secret*.

The rule (Level Pair) says that the level of a pair is the level of its components. Both components must have the same level; when we pair a term of level *Public* and one of level *Secret*, we need to regard them both as having level *Any*. Thus, the rule (Level Pair) loses a little bit of typing information; it would be interesting to explore a richer, more “structural” typing system that would avoid this loss.

The rule (Level Encryption *Public*) says that k pieces of data of the same level T can be encrypted under a key of level *Public*, with a resulting ciphertext of level T. The rule (Level Encryption *Secret*) imposes more restrictions for encryption under keys of level *Secret*, because the resulting ciphertext is declassified to level *Public*. These restrictions enforce a particular format for the levels of the contents and the use of a confounder, as explained in Section 2. One could relax this rule somewhat, considering also the case where the resulting ciphertext is given a level other than *Public*; the present rule strikes a reasonable balance between simplicity and flexibility. Finally, note that there is no rule for encryption for the case where N is a term of level *Any*. If N is a term of level *Any*, and it is not known whether it is of level *Public* or *Secret*, then N cannot be used as a key.

4.3 Processes

The rules for processes are:

(Level Output *Public*)

$$\frac{E \vdash M : \mathit{Public} \qquad E \vdash M_1 : \mathit{Public} \ \ldots \ E \vdash M_k : \mathit{Public} \qquad E \vdash P : \mathit{Ok}}{E \vdash \overline{M}\langle M_1, \ldots, M_k\rangle.P : \mathit{Ok}}$$

(Level Output *Secret*)

$$\frac{E \vdash M : \mathit{Secret} \qquad E \vdash M_1 : \mathit{Secret} \qquad E \vdash M_2 : \mathit{Any} \qquad E \vdash M_3 : \mathit{Public} \qquad E \vdash P : \mathit{Ok}}{E \vdash \overline{M}\langle M_1, M_2, M_3\rangle.P : \mathit{Ok}}$$

(Level Input *Public*)

$$\frac{E \vdash M : \mathit{Public} \qquad E, x_1 : \mathit{Public}, \ldots, x_k : \mathit{Public} \vdash P : \mathit{Ok}}{E \vdash M(x_1, \ldots, x_k).P : \mathit{Ok}}$$

(Level Input *Secret*)

$$\frac{E \vdash M : \mathit{Secret} \qquad E, x_1 : \mathit{Secret}, x_2 : \mathit{Any}, x_3 : \mathit{Public} \vdash P : \mathit{Ok}}{E \vdash M(x_1, x_2, x_3).P : \mathit{Ok}}$$

(Level Nil)

$$\frac{\vdash E \text{ well-formed}}{E \vdash \mathbf{0} : \mathit{Ok}}$$

(Level Parallel)

$$\frac{E \vdash P : \mathit{Ok} \qquad E \vdash Q : \mathit{Ok}}{E \vdash P \mid Q : \mathit{Ok}}$$

(Level Replication)

$$\frac{E \vdash P : \mathit{Ok}}{E \vdash !P : \mathit{Ok}}$$

(Level Restriction)

$$\frac{E, n : T :: L \vdash P : \mathit{Ok}}{E \vdash (\nu n)P : \mathit{Ok}}$$

(Level Match) for $T, R \in \{\mathit{Public}, \mathit{Secret}\}$

$$\frac{E \vdash M : T \qquad E \vdash N : R \qquad E \vdash P : \mathit{Ok}}{E \vdash [M \ \mathit{is} \ N] \, P : \mathit{Ok}}$$

(Level Pair Splitting) for $T \in \{\mathit{Public}, \mathit{Secret}\}$

$$\frac{E \vdash M : T \qquad E, x : T, y : T \vdash P : \mathit{Ok}}{E \vdash \mathit{let} \ (x, y) = M \ \mathit{in} \ P : \mathit{Ok}}$$

(Level Integer Case) for $T \in \{\mathit{Public}, \mathit{Secret}\}$

$$\frac{E \vdash M : T \qquad E \vdash P : \mathit{Ok} \qquad E, x : T \vdash Q : \mathit{Ok}}{E \vdash \mathit{case} \ M \ \mathit{of} \ 0 : P \ \mathit{suc}(x) : Q : \mathit{Ok}}$$

(Level Decryption *Public*) for $T \in \{\mathit{Public}, \mathit{Secret}\}$

$$\frac{E \vdash L : T \qquad E \vdash N : \mathit{Public} \qquad E, x_1 : T, \ldots, x_k : T \vdash P : \mathit{Ok}}{E \vdash \mathit{case} \ L \ \mathit{of} \ \{x_1, \ldots, x_k\}_N \ \mathit{in} \ P : \mathit{Ok}}$$

(Level Decryption *Secret*) for $T \in \{\mathit{Public}, \mathit{Secret}\}$

$$\frac{\begin{array}{c} E \vdash L : T \qquad E \vdash N : \mathit{Secret} \\ E, x_1 : \mathit{Secret}, x_2 : \mathit{Any}, x_3 : \mathit{Public}, x_4 : \mathit{Any} \vdash P : \mathit{Ok} \end{array}}{E \vdash \mathit{case} \ L \ \mathit{of} \ \{x_1, x_2, x_3, x_4\}_N \ \mathit{in} \ P : \mathit{Ok}}$$

There are two rules for output and two rules for input. The rule (Level Output *Public*) says that terms of level *Public* may be sent on a channel of level *Public*. The rule (Level Output *Secret*) says that terms of all levels may be sent on a channel of level *Secret*—provided this is done according to the format described in Section 2. The two rules for input, (Level Input *Public*) and (Level Input *Secret*), match these rules for output. In (Level Input *Public*) all inputs are assumed to be of level *Public*, while in (Level Input *Secret*) the levels of the inputs are deduced from their position, as allowed by the format of messages on channels of level *Secret*. If M is a term of level *Any*, and it is not known whether it is of level *Public* or *Secret*, then M cannot be used as a channel.

The rules for nil, for parallel composition, for replication, and for restriction are routine. In the rule (Level Restriction), the name n being bound is associated with an arbitrary term L for which it can be used as a confounder.

The rule (Level Match) enables us to compare any two terms of levels *Public* or *Secret*. Terms of level *Any* are excluded in order to prevent implicit flows, as discussed in Section 2. It may be a little surprising that terms of level *Secret* are allowed in this rule, because this may seem to permit an implicit flow. However, the generality of the rule (Level Match) does not present an obstacle to our results.

The rule (Level Pair Splitting) enables us to try to break a term of level *Public* or *Secret* into two components, each assumed to be of the same level as the original term. The case where the original term is known only to be of level *Any* is disallowed; if it were allowed, this rule would permit leaking whether the term is in fact a pair.

Similarly, the rule (Level Integer Case) enables to examine whether a term is 0 or a successor term, and to branch on the result. In the successor case, the variable x represents the predecessor of the term being examined, which is assumed to be of the same level as the term. As in (Level Pair Splitting), the term should not be of level *Any*, but it may be of level *Secret*. If names were numbers, then repeated applications of the rule (Level Integer Case) would enable us to publish a secret key in unary. Formally, this leak cannot happen because names are not numbers. The practical meaning of this small formal miracle is debatable; it may suggest that a model more concrete than the spi calculus would be worth investigating.

Finally, there are two rules for decryption. The rule (Level Decryption *Public*) handles the case where the decryption key is of level *Public*, while the rule (Level Decryption *Secret*) handles the case where the decryption key is of level *Secret*. These rules are analogous to the corresponding rules for input. There is no rule for the case of a key of level *Any*.

5 What Typing Guarantees

The goal of this section is to prove that if a process typechecks then it does not leak the values of parameters of level *Any*. More precisely, our main theorem says that if only variables of level *Any* and only names of level *Public* are in the domain of the environment E, if σ and σ' are two substitutions of values for the variables in E, and if P typechecks (that is, $E \vdash P : Ok$ can be proved), then $P\sigma$ and $P\sigma'$ are testing equivalent. This conclusion means that an observer cannot distinguish $P\sigma$ and $P\sigma'$, so it cannot detect the difference in the values for the variables.

In order to prove this result, we develop a number of propositions and lemmas that analyze the typing system and characterize the possible behaviors of processes that typecheck.

5.1 Typing for Concretions and Abstractions

The first step in our proofs is to extend the typing rules to concretions and abstractions:

(Level Concretion *Public*)

$$\frac{\begin{array}{c} E, m_1 : T_1 :: L_1, \ldots, m_l : T_l :: L_l \vdash M_1 : \mathit{Public} \\ \vdots \\ E, m_1 : T_1 :: L_1, \ldots, m_l : T_l :: L_l \vdash M_k : \mathit{Public} \\ E, m_1 : T_1 :: L_1, \ldots, m_l : T_l :: L_l \vdash P : \mathit{Ok} \end{array}}{E \vdash (\nu m_1, \ldots, m_l)\langle M_1, \ldots, M_k\rangle P : \mathit{OkCPublic}}$$

(Level Concretion *Secret*)

$$\frac{\begin{array}{c} E, m_1 : T_1 :: L_1, \ldots, m_l : T_l :: L_l \vdash M_1 : \mathit{Secret} \\ E, m_1 : T_1 :: L_1, \ldots, m_l : T_l :: L_l \vdash M_2 : \mathit{Any} \\ E, m_1 : T_1 :: L_1, \ldots, m_l : T_l :: L_l \vdash M_3 : \mathit{Public} \\ E, m_1 : T_1 :: L_1, \ldots, m_l : T_l :: L_l \vdash P : \mathit{Ok} \end{array}}{E \vdash (\nu m_1, \ldots, m_l)\langle M_1, M_2, M_3\rangle P : \mathit{OkCSecret}}$$

(Level Abstraction *Public*)

$$\frac{E, x_1 : \mathit{Public}, \ldots, x_k : \mathit{Public} \vdash P : \mathit{Ok}}{E \vdash (x_1, \ldots, x_k)P : \mathit{OkAPublic}}$$

(Level Abstraction *Secret*)

$$\frac{E, x_1 : \mathit{Secret}, x_2 : \mathit{Any}, x_3 : \mathit{Public} \vdash P : \mathit{Ok}}{E \vdash (x_1, x_2, x_3)P : \mathit{OkASecret}}$$

These rules should be reminiscent of corresponding rules for output and input.

5.2 Auxiliary Propositions

Next we obtain several auxiliary results. The first of them, Proposition 2, is a formal counterpart of the discussion of Section 2.1: this proposition shows that, given a suitable environment, a closed process P that we may construe as an attacker always typechecks.

Proposition 2. *Assume that* $\vdash E$ well-formed*, that* $\mathit{dom}(E)$ *does not contain any variables, and that all the names in* $\mathit{dom}(E)$ *are of level Public.*

- *If* M *is a closed term and* $\mathit{fn}(M) \subseteq \mathit{dom}(E)$*, then* $E \vdash M : \mathit{Public}$*.*
- *If* P *is a closed process and* $\mathit{fn}(P) \subseteq \mathit{dom}(E)$*, then* $E \vdash P : \mathit{Ok}$*.*

Proof. We prove a more general property, allowing variables to occur in E, M, and P. We consider an environment E such that $\vdash E$ well-formed, where the levels of names and variables are all *Public*. We prove that:

- If M is a term with $\mathit{fn}(M) \cup \mathit{fv}(M) \subseteq \mathit{dom}(E)$, then $E \vdash M : \mathit{Public}$.

– If P is a process with $fn(P) \cup fv(P) \subseteq dom(E)$, then $E \vdash P : Ok$.

The former of these facts is obtained by a direct induction on the structure of M (using (Level Encryption *Public*) for terms of the form $\{M_1, \ldots, M_k\}_N$). The latter of these facts is then obtained by a direct induction on the structure of P (using (Level Output *Public*), (Level Input *Public*), and (Level Decryption *Public*)). $\square$

Proposition 3 is fairly standard; it says that anything that can be proved in a given environment can also be proved after adding assumptions to the environment.

Proposition 3. *Assume that* $\vdash E, E', E''$ well-formed.

– *If* $E, E'' \vdash M : T$ *then* $E, E', E'' \vdash M : T$.
– *If* $E, E'' \vdash P : Ok$ *then* $E, E', E'' \vdash P : Ok$.

Proof. This property is proved by induction on the derivations of $E, E'' \vdash M : T$ and $E, E'' \vdash P : T$. $\square$

Proposition 4 enables to reorder an environment, moving the declaration of a name past the declarations of some variables.

Proposition 4. *Let* E_1 *be* $E, n : T_0 :: L, x_1 : T_1, \ldots, x_k : T_k, E'$ *and let* E_2 *be* $E, x_1 : T_1, \ldots, x_k : T_k, n : T_0 :: L, E'$.

– *If* $\vdash E_1$ well-formed *then* $\vdash E_2$ well-formed.
– *If* $E_1 \vdash M : T$ *then* $E_2 \vdash M : T$.
– *If* $E_1 \vdash P : Ok$ *then* $E_2 \vdash P : Ok$.

Proof. The proof is by induction on the derivations of $\vdash E_1$ well-formed, $E_1 \vdash M : T$, and $E_1 \vdash P : Ok$. $\square$

Note that the converse of this proposition cannot be true because, under the hypotheses of the converse, L could contain free occurrences of $x_1, \ldots, x_k$.

The next proposition says that the levels *Secret* and *Public* are mutually exclusive.

Proposition 5. *If* $E \vdash M : Secret$ *then it is not the case that* $E \vdash M : Public$.

Proof. We assume that $E \vdash M : Secret$ and $E \vdash M : Public$ both hold, and derive a contradiction, by induction on the size of the two derivations of $E \vdash M : Secret$ and $E \vdash M : Public$. The only interesting case is that where M has the form $\{M_1, \ldots, M_k\}_N$. The term $\{M_1, \ldots, M_k\}_N$ could have levels *Secret* and *Public* at once only if N did as well (and by application of the two rules for encryption); the induction hypothesis yields the expected contradiction. $\square$

The remaining auxiliary results all concern substitutions. The first of them is a standard substitution lemma.

Proposition 6. *Assume that* $E \vdash N : T$.

- *If* $\vdash E, x : T, E'$ well-formed *then* $\vdash E, E'[N/x]$ well-formed.
- *If* $E, x : T, E' \vdash M : R$ *then* $E, E'[N/x] \vdash M[N/x] : R$.
- *If* $E, x : T, E' \vdash P : Ok$ *then* $E, E'[N/x] \vdash P[N/x] : Ok$.

Proof. The proof is by a joint induction on the derivations of the judgments $\vdash E, x : T, E'$ well-formed, $E, x : T, E' \vdash M : R$, and $E, x : T, E' \vdash P : Ok$. In the case of the rule (Level Encryption *Secret*), it is important to note that if the confounder n appears in E then x cannot occur in the term being formed. □

In general, a substitution is a partial function from the set of variables to the set of terms. We write $dom(\sigma)$ for the domain of the substitution σ.

Proposition 7. *Given an environment* E*, suppose that only variables of level* Any *in* E *are in* $dom(\sigma)$*, and that either* $E \vdash L : Public$ *or* $E \vdash L : Secret$.

- *If* $L\sigma$ *is a variable then* L *is the same variable.*
- *If* $L\sigma$ *is a name then* L *is the same name.*
- *If* $L\sigma$ *is of the form* (M, N) *then* L *is of the same form.*
- *If* $L\sigma$ *is* 0 *then* L *is* 0.
- *If* $L\sigma$ *is of the form* $suc(M)$ *then* L *is of the same form.*
- *If* $L\sigma$ *is of the form* $\{M_1, \ldots, M_k\}_N$ *then* L *is of the same form.*

Proof. This follows from the fact that L cannot be a variable in $dom(\sigma)$, since this domain consists of variables of level Any. □

Proposition 8. *Given an environment* E*, suppose that only variables of level* Any *in* E *are in* $dom(\sigma)$*. Suppose further that* $E \vdash M : T$ *and* $E \vdash N : R$*, where* $T, R \in \{Public, Secret\}$*. If* $M\sigma = N\sigma$ *then* $M = N$.

Proof. The proof is by induction on the derivation of $E \vdash M : T$. The rule (Level Subsumption) can be applied only trivially as the last rule of this derivation since $T \in \{Public, Secret\}$. There is one case for each of the remaining rules for typechecking terms.

- If the last rule of the derivation of $E \vdash M : T$ is (Level Variable), then M is a variable, but not one in $dom(\sigma)$ since $dom(\sigma)$ consists of variables of level Any, so $M\sigma = M$. Therefore, $M\sigma = N\sigma$ implies that $M = N$ by Proposition 7.
- If the last rule of the derivation of $E \vdash M : T$ is (Level Name) or (Level Zero), then M is a name or 0, so $M\sigma = M$. Therefore, $M\sigma = N\sigma$ implies that $M = N$ by Proposition 7.
- The cases of (Level Successor) and (Level Pair) are by easy applications of Proposition 7 and the induction hypothesis.

- The two remaining cases are for M of the form $\{M_1, \ldots, M_k\}_L$. We have $E \vdash L : \mathit{Public}$ or $E \vdash L : \mathit{Secret}$, depending on whether the derivation of $E \vdash M : T$ finishes with an application of (Level Encryption *Public*) or an application of (Level Encryption *Secret*). In both cases, Proposition 7 implies that N has the form $\{N_1, \ldots, N_k\}_{L'}$, with $L\sigma = L'\sigma$ and either $E \vdash L' : \mathit{Public}$ or $E \vdash L' : \mathit{Secret}$ depending on whether the derivation of $E \vdash N : R$ finishes with an application of (Level Encryption *Public*) or an application of (Level Encryption *Secret*). By induction hypothesis, we obtain $L = L'$. By Proposition 5, the derivations of $E \vdash M : T$ and $E \vdash N : R$ both finish with applications of the same rule.
  - If this rule is (Level Encryption *Public*), then we have $E \vdash M_1 : \mathit{Public}$, $\ldots$, $E \vdash M_k : \mathit{Public}$, $E \vdash N_1 : \mathit{Public}$, $\ldots$, $E \vdash N_k : \mathit{Public}$ as hypotheses to the applications of the rule. Since $M\sigma = N\sigma$, we have $M_1\sigma = N_1\sigma$, $\ldots$, $M_k\sigma = N_k\sigma$. By induction hypothesis, we obtain $M_1 = N_1$, $\ldots$, $M_k = N_k$, and therefore $M = N$.
  - If this rule is (Level Encryption *Secret*), then $k = 4$, and M_4 is a name m with $m : T' :: M$ in E for some level T', and N_4 is a name n with $n : T'' :: N$ in E for some level T''. Since $M\sigma = N\sigma$, we have $m = n$. Since a name may be declared in an environment at most once, we conclude that $M = N$.

$\square$

5.3 Lemmas on Commitment and Simulation

The main lemma of this section relates the typing system with the commitment relation. We write $E \vdash \sigma$ when, for every $x \in \mathit{dom}(E)$, $\sigma(x)$ is a closed term such that $\mathit{fn}(\sigma(x)) \subseteq \mathit{dom}(E)$.

Lemma 9. *Assume that:*

(1) $E \vdash P : \mathit{Ok}$,
(2) $E \vdash \sigma$,
(3) *all variables in* $\mathit{dom}(E)$ *are of level* Any.

Then:

(1) *If* $P\sigma > Q'$ *then there is a process* Q *such that*
  - $Q' = Q\sigma$,
  - $E \vdash Q : \mathit{Ok}$,
  - $P\sigma' > Q\sigma'$ *whenever* $E \vdash \sigma'$.

(2) *If* $P\sigma \xrightarrow{\tau} Q'$ *then there is a process* Q *such that*
  - $Q' = Q\sigma$,
  - $E \vdash Q : \mathit{Ok}$,
  - $P\sigma' \xrightarrow{\tau} Q\sigma'$ *whenever* $E \vdash \sigma'$.

(3) *If* $P\sigma \xrightarrow{\overline{m}} A'$ *then either* $E \vdash m : \mathit{Public}$ *or* $E \vdash m : \mathit{Secret}$, *and there is a concretion* A *such that*

- $A' = A\sigma$,
- *either* $E \vdash A : OkCPublic$ *or* $E \vdash A : OkCSecret$ *(depending on whether* $E \vdash m : Public$ *or* $E \vdash m : Secret$*),*
- $P\sigma' \xrightarrow{\overline{m}} A\sigma'$ *whenever* $E \vdash \sigma'$.

(4) *If* $P\sigma \xrightarrow{m} A'$ *then either* $E \vdash m : Public$ *or* $E \vdash m : Secret$*, and there is an abstraction* A *such that*
- $A' = A\sigma$,
- *either* $E \vdash A : OkAPublic$ *or* $E \vdash A : OkASecret$ *(depending on whether* $E \vdash m : Public$ *or* $E \vdash m : Secret$*),*
- $P\sigma' \xrightarrow{m} A\sigma'$ *whenever* $E \vdash \sigma'$.

Proof. The argument for this lemma is quite long, but not surprising, given the auxiliary propositions. It consists of one case for each of the axioms for the reduction relation and one case for each of the rules for the commitment relation. We therefore omit the details of this argument. □

Lemma 10. *Given an environment* E*, suppose that all the variables in* $dom(E)$ *are of level Any. Suppose further that* $E \vdash \sigma$ *and* $E \vdash \sigma'$*. Then the relation*

$$\{(P\sigma, P\sigma') \mid E \vdash P : Ok\}$$

is a strict barbed bisimulation.

Proof. First we consider any commitment $P\sigma \xrightarrow{\alpha} A'$, where α is a barb. By Lemma 9, there is an agent A such that $P\sigma' \xrightarrow{\alpha} A\sigma'$.

Next, we consider any commitment $P\sigma \xrightarrow{\tau} Q'$. By Lemma 9, there is a process Q such that $E \vdash Q : Ok$, $Q' = Q\sigma$, and $P\sigma' \xrightarrow{\tau} Q\sigma'$. Thus, any τ step of $P\sigma$ may be matched by $P\sigma'$.

Therefore, the relation $\{(P\sigma, P\sigma') \mid E \vdash P : Ok\}$ is a strict barbed simulation. By symmetry, it is a strict barbed bisimulation. □

5.4 Main Theorem

Finally, we obtain the theorem described at the start of Section 5:

Theorem 11. *Given an environment* E*, suppose that only variables of level Any and only names of level Public are in* $dom(E)$*. Suppose further that* $E \vdash \sigma$ *and* $E \vdash \sigma'$*. If* $E \vdash P : Ok$ *then* $P\sigma \simeq P\sigma'$.

Proof. According to Lemma 1, it suffices to show that for every closed process Q there exists a strict barbed bisimulation that relates $P\sigma \mid Q$ and $P\sigma' \mid Q$. Since Q is closed, $P\sigma \mid Q$ equals $(P \mid Q)\sigma$, and $P\sigma' \mid Q$ equals $(P \mid Q)\sigma'$.

We construct an extension of E with the names that appear free in Q but are not in $dom(E)$. For each such name n, we add $n : Public$ to E. Let us call E^* the resulting environment. By Propositions 2 and 3, we obtain $E^* \vdash Q : Ok$. Also by Proposition 3, we obtain $E^* \vdash P : Ok$. Combining these two results, rule (Level Parallel) yields $E^* \vdash (P \mid Q) : Ok$.

Finally, Lemma 10 yields the desired result. □

Note that this theorem would not hold if P could have free occurrences of names of level *Secret*. Such occurrences are ruled out by the hypotheses that only names of level *Public* are in E and that $E \vdash P : Ok$.

6 Examples

In order to illustrate the use of our typing rules, we consider as examples two protocols for key exchange and secure communication.

In both cases, we can typecheck the protocols. As corollaries, we obtain that the secrecy of certain messages is protected. These corollaries should not be surprising. However, they would be much harder to prove in the spi calculus without the rules developed in this paper, from first principles.

Analogues of our corollaries might be provable in other formal systems. Surprisingly, there do not seem to be any formal proofs of this sort in the literature. In some methods, one may be able to show that the messages in question are not among the terms that the attacker obtains when the protocol runs. However, this result is only an approximation to the corollaries, as it does not rule out that the attacker could at least deduce whether the messages are even numbers or odd numbers, for example. The corollaries exclude this possibility.

Analogues of our corollaries can perhaps be established in informal but rigorous models (see for example [BR95]). These models are rather accurate, as in particular they can take into account issues of probability and complexity. Unfortunately, proofs in these models remain much more difficult than typechecking.

6.1 A First Example

The first protocol is similar in structure to the Wide Mouthed Frog protocol [BAN89]. Informally, this protocol is:

Message 1	$A \to S:$	A	on c_S
Message 2	$S \to A:$	N_S	on c_A
Message 3	$A \to S:$	$\{K_{AB}, *, (A, A, B, N_S), C_A\}_{K_{AS}}$	on c_S
Message 4	$S \to B:$	$*$	on c_B
Message 5	$B \to S:$	N_B	on c_S
Message 6	$S \to B:$	$\{K_{AB}, *, (S, A, B, N_B), C_S\}_{K_{SB}}$	on c_B
Message 7	$A \to B:$	$\{*, M, *, C'_A\}_{K_{AB}}$	on c_B
Message 8	$B \to ?:$	A, B	on d_B

In this protocol, A and B are two clients and S is a server. The channels c_S, c_A, c_B, and d_B are public. The keys K_{AS} and K_{SB} are secret keys for communication with the server, while K_{AB} is a new key for communication from A to B. The message M is intended to be secret. Both N_S and N_B are nonces (used to prove timeliness); $*$ is an arbitrary message of appropriate level (not necessarily the same for all occurrences of $*$); and C_A, C'_A, and C_S are confounders. In Messages 1 and 2, A requests and receives a nonce challenge from S. In Messages 4 and 5, S requests and receives a nonce challenge from B. In Message 3, A provides the key K_{AB} to S, which passes it on to B in Message 6. In Message 7, A uses K_{AB} for sending M. On receipt of this message from A, the recipient B outputs the names of A and B on a public channel d_B, in Message 8. It is not important to specify who receives this last message, which we include only in order to illustrate that B is allowed to react.

We can express this protocol in the spi calculus, much as in the earlier work on the spi calculus [AG97a] but with attention to the requirements of typing. The definition is for a given set of messages $M_1, \ldots, M_m$ with source addresses $i_1, \ldots, i_m$ and destination addresses $j_1, \ldots, j_m$, respectively. These addresses are natural numbers in the range $1..n$; they indicate who plays the role of A and who plays the role of B in each run of the protocol. In addition, S is an address. For each address i, there are channels c_i and d_i. We write $\underline{i}$ for the term representing i, and simply write S for the term representing S. In the definition of $\mathit{Send}_{i,j}$, the variable z corresponds to the message M; we write $\mathit{Send}_{i,j}(M)$ for $\mathit{Send}_{i,j}[M/z]$.

$$
\begin{array}{rcl}
\mathit{Send}_{i,j} & \triangleq & \overline{c_S}\langle \underline{i} \rangle \mid \\
& & c_i(x_{nonce}). \\
& & (\nu K)((\nu C_A)\overline{c_S}\langle \{K, *, (\underline{i}, \underline{i}, \underline{j}, x_{nonce}), C_A\}_{K_{iS}} \rangle \mid \\
& & \qquad (\nu C'_A)\overline{c_j}\langle \{*, z, *, C'_A\}_K \rangle) \\
S & \triangleq & c_S(x_A).\prod_{i \in 1..n}[x_A \ \mathit{is}\ \underline{i}]\ (\nu N_S)(\overline{c_i}\langle N_S \rangle \mid \\
& & c_S(x_{cipher}). \\
& & \mathit{case}\ x_{cipher}\ \mathit{of}\ \{x_{key}, x_a, x_p, x_{cnf}\}_{K_{iS}}\ \mathit{in} \\
& & \mathit{let}\ (y_A, z_A, x_B, x_{nonce}) = x_p\ \mathit{in} \\
& & \prod_{j \in 1..n}[y_A\ \mathit{is}\ \underline{i}]\ [z_A\ \mathit{is}\ \underline{i}]\ [x_B\ \mathit{is}\ \underline{j}]\ [x_{nonce}\ \mathit{is}\ N_S] \\
& & (\overline{c_j}\langle * \rangle \mid \\
& & \ c_S(y_{nonce}). \\
& & \ (\nu C_S)\overline{c_j}\langle \{x_{key}, *, (S, \underline{i}, \underline{j}, y_{nonce}), C_S\}_{K_{Sj}} \rangle)) \\
\mathit{Recv}_j & \triangleq & c_j(w).(\nu N_B)(\overline{c_S}\langle N_B \rangle \mid \\
& & c_j(y_{cipher}). \\
& & \mathit{case}\ y_{cipher}\ \mathit{of}\ \{x_{key}, y_a, y_p, y_{cnf}\}_{K_{Sj}}\ \mathit{in} \\
& & \mathit{let}\ (x_S, x_A, x_B, y_{nonce}) = y_p\ \mathit{in} \\
& & \prod_{i \in 1..n}[x_S\ \mathit{is}\ S]\ [x_A\ \mathit{is}\ \underline{i}]\ [x_B\ \mathit{is}\ \underline{j}]\ [y_{nonce}\ \mathit{is}\ N_B] \\
& & c_j(z_{cipher}). \\
& & \mathit{case}\ z_{cipher}\ \mathit{of}\ \{z_s, z_a, z_p, z_{cnf}\}_{x_{key}}\ \mathit{in} \\
& & \overline{d_j}\langle \underline{i}, \underline{j} \rangle) \\
\mathit{Sys} & \triangleq & (\nu K_{1S}) \ldots (\nu K_{nS}) \\
& & (\nu K_{S1}) \ldots (\nu K_{Sn}) \\
& & (\mathit{Send}_{i_1,j_1}(M_1) \mid \ldots \mid \mathit{Send}_{i_m,j_m}(M_m) \mid \\
& & \ !S \mid \\
& & \ !\mathit{Recv}_1 \mid \ldots \mid !\mathit{Recv}_n)
\end{array}
$$

The next proposition says that this protocol typechecks.

Proposition 12. *Let E be the environment*

$$c_S : Public, c_1 : Public, \ldots, c_n : Public, d_1 : Public, \ldots, d_n : Public,$$
$$z_1 : Any, \ldots, z_m : Any$$

Let i_k and j_k be any fixed numbers in $1..n$, for $k \in 1..m$. Let M_k be z_k, for $k \in 1..m$. Then $E \vdash Sys : Ok$.

Proof. In order to indicate how the process *Sys* typechecks, we annotate its bound names and variables with their levels, as they are introduced; we also annotate confounders with the terms in which they are used. For $K_{iS} : Secret$, $K_{Sj} : Secret$, and $z : Any$, we define:

$$\begin{array}{l}
Send_{i,j} \quad \triangleq \\
\quad \overline{c_S}\langle \underline{i} \rangle \mid \\
\quad c_i(x_{nonce} : Public). \\
\quad (\nu K : Secret) \\
\quad ((\nu C_A : Secret :: \{K, *, (\underline{i}, \underline{i}, \underline{j}, x_{nonce}), C_A\}_{K_{iS}}) \\
\qquad \overline{c_S}\langle \{K, *, (\underline{i}, \underline{i}, \underline{j}, x_{nonce}), C_A\}_{K_{iS}} \rangle \mid \\
\quad\; (\nu C'_A : Secret :: \{*, z, *, C'_A\}_K) \\
\qquad \overline{c_j}\langle \{*, z, *, C'_A\}_K \rangle) \\
S \quad \triangleq \\
\quad c_S(x_A : Public). \prod_{i \in 1..n} [x_A \ is \ \underline{i}] \ (\nu N_S : Public)(\overline{c_i}\langle N_S \rangle \mid \\
\quad c_S(x_{cipher} : Public). \\
\quad case \ x_{cipher} \ of \ \{x_{key} : Secret, x_a : Any, x_p : Public, x_{cnf} : Any\}_{K_{iS}} \ in \\
\quad let \ (y_A : Public, z_A : Public, x_B : Public, x_{nonce} : Public) = x_p \ in \\
\quad \prod_{j \in 1..n} [y_A \ is \ \underline{i}] \ [z_A \ is \ \underline{i}] \ [x_B \ is \ \underline{j}] \ [x_{nonce} \ is \ N_S] \\
\quad (\overline{c_j}\langle * \rangle \mid \\
\quad\; c_S(y_{nonce} : Public). \\
\quad\; (\nu C_S : Secret :: \{x_{key}, *, (S, \underline{i}, \underline{j}, y_{nonce}), C_S\}_{K_{Sj}}) \\
\qquad \overline{c_j}\langle \{x_{key}, *, (S, \underline{i}, \underline{j}, y_{nonce}), C_S\}_{K_{Sj}} \rangle)) \\
Recv_j \quad \triangleq \\
\quad c_j(w : Public).(\nu N_B : Public)(\overline{c_S}\langle N_B \rangle \mid \\
\quad c_j(y_{cipher} : Public). \\
\quad case \ y_{cipher} \ of \ \{x_{key} : Secret, y_a : Any, y_p : Public, y_{cnf} : Any\}_{K_{Sj}} \ in \\
\quad let \ (x_S : Public, x_A : Public, x_B : Public, y_{nonce} : Public) = y_p \ in \\
\quad \prod_{i \in 1..n} [x_S \ is \ S] \ [x_A \ is \ \underline{i}] \ [x_B \ is \ \underline{j}] \ [y_{nonce} \ is \ N_B] \\
\quad c_j(z_{cipher} : Public). \\
\quad case \ z_{cipher} \ of \ \{z_s : Secret, z_a : Any, z_p : Public, z_{cnf} : Any\}_{x_{key}} \ in \\
\quad \overline{d_j}\langle \underline{i}, \underline{j} \rangle)
\end{array}$$

Finally, in the given environment E, we set:

$$\begin{array}{rcl}
Sys & \triangleq & (\nu K_{1S} : Secret) \ldots (\nu K_{nS} : Secret) \\
& & (\nu K_{S1} : Secret) \ldots (\nu K_{Sn} : Secret) \\
& & (Send_{i_1,j_1}(z_1) \mid \ldots \mid Send_{i_m,j_m}(z_m) \mid \\
& & \;!S \mid \\
& & \;!Recv_1 \mid \ldots \mid !Recv_n)
\end{array}$$

writing $Send_{i_k,j_k}(z_k)$ for $Send_{i_k,j_k}[z_k/z]$. □

As a consequence of the typechecking, we obtain that the protocol does not reveal the message M from A. This conclusion is stated in the following corollary, where for simplicity we restrict attention to the case where M is a numeral. (A numeral is one of the terms 0, $suc(0)$, $suc(suc(0))$, ...)

Corollary 13. *Let i_k and j_k be any fixed numbers in $1..n$, for $k \in 1..m$. Let Sys1 and Sys2 be two versions of Sys where the terms M_k are arbitrary numerals, for $k \in 1..m$. Then $Sys1 \simeq Sys2$.*

Proof. This is an immediate consequence of Proposition 12 and Theorem 11. □

6.2 A Second Example

In the second example, the principal A transmits the secret message M under a key generated by the server S. This example is interesting because it brings up an issue of trust: A trusts S to provide a key appropriate for the transmission of M.

Message 1	$A \to B:$	A, N_A	on c_B
Message 2	$B \to S:$	A, N_A, B, N_B	on c_S
Message 3	$S \to A:$	$\{K_{AB}, *, (A, B, N_A), C_S\}_{K_{SA}}$	on c_A
Message 4	$S \to B:$	$\{K_{AB}, *, (A, B, N_B), C'_S\}_{K_{SB}}$	on c_B
Message 5	$A \to B:$	$\{*, M, *, C_A\}_{K_{AB}}$	on c_B
Message 6	$B \to ?:$	A, B	on d_B

Again, A and B are two clients and S is a server, and c_S, c_A, c_B, and d_B are public channels. The keys K_{SA} and K_{SB} are secret keys for communication from the server to A and B, while K_{AB} is the new key for communication between A and B. Both N_A and N_B are nonces, and C_S, C'_S, and C_A are confounders.

We write this example in the spi calculus in much the same style as the first example. The definition is for a given set of messages $M_1, \ldots, M_m$ with source addresses $i_1, \ldots, i_m$ and destination addresses $j_1, \ldots, j_m$, respectively.

$$
\begin{array}{rcl}
Send_{i,j} & \triangleq & (\nu N_A) \\
& & (\overline{c_j}\langle \underline{i}, N_A\rangle \mid \\
& & \;c_i(x_{cipher}). \\
& & \;case\ x_{cipher}\ of\ \{x_{key}, x_a, x_p, x_{cnf}\}_{K_{Si}}\ in \\
& & \;let\ (x_A, y_B, x_{nonce}) = x_p\ in \\
& & \;[x_A\ is\ \underline{i}]\ [y_B\ is\ \underline{j}]\ [x_{nonce}\ is\ N_A] \\
& & \;(\nu C_A)\overline{c_j}\langle\{*, z, *, C_A\}_{x_{key}}\rangle) \\
S & \triangleq & c_S(x_A, x_{nonce}, y_B, y_{nonce}). \\
& & \prod_{i\in 1..n}[x_A\ is\ \underline{i}]\ \prod_{j \in 1..n}[y_B\ is\ \underline{j}] \\
& & (\nu K) \\
& & ((\nu C_S)\overline{c_i}\langle\{K, *, (\underline{i}, \underline{j}, x_{nonce}), C_S\}_{K_{Si}}\rangle \mid \\
& & \;(\nu C'_S)\overline{c_j}\langle\{K, *, (\underline{i}, \underline{j}, y_{nonce}), C'_S\}_{K_{Sj}}\rangle)
\end{array}
$$

$$
\begin{array}{rcl}
Recv_j & \triangleq & c_j(x_A, x_{nonce}). \\
& & \prod_{i \in 1..n} [x_A \textit{ is } \underline{i}] \\
& & (\nu N_B) \\
& & (\overline{c_S}\langle \underline{i}, x_{nonce}, \underline{j}, N_B \rangle \mid \\
& & \;c_j(y_{cipher}). \\
& & \;\textit{case } y_{cipher} \textit{ of } \{x_{key}, y_a, y_p, y_{cnf}\}_{K_{Sj}} \textit{ in} \\
& & \;\textit{let } (x_A, y_B, y_{nonce}) = y_p \textit{ in} \\
& & \;[x_A \textit{ is } \underline{i}]\ [y_B \textit{ is } \underline{j}]\ [y_{nonce} \textit{ is } N_B] \\
& & \;c_j(z_{cipher}). \\
& & \;\textit{case } z_{cipher} \textit{ of } \{z_s, z_a, z_p, z_{cnf}\}_{x_{key}} \textit{ in} \\
& & \;\overline{d_j}\langle \underline{i}, \underline{j} \rangle) \\
Sys & \triangleq & (\nu K_{S1}) \ldots (\nu K_{Sn}) \\
& & (Send_{i_1,j_1}(M_1) \mid \ldots \mid Send_{i_m,j_m}(M_m) \mid \\
& & !S \mid \\
& & !Recv_1 \mid \ldots \mid !Recv_n)
\end{array}
$$

The next proposition and corollary are the analogues of Proposition 12 and Corollary 13, respectively. We omit their proofs.

Proposition 14. *Let E be the environment*

$$
\begin{array}{l}
c_S : \textit{Public}, c_1 : \textit{Public}, \ldots, c_n : \textit{Public}, d_1 : \textit{Public}, \ldots, d_n : \textit{Public}, \\
z_1 : \textit{Any}, \ldots, z_m : \textit{Any}
\end{array}
$$

Let i_k and j_k be any fixed numbers in $1..n$, for $k \in 1..m$. Let M_k be z_k, for $k \in 1..m$. Then $E \vdash Sys : Ok$.

Corollary 15. *Let i_k and j_k be any fixed numbers in $1..n$, for $k \in 1..m$. Let Sys1 and Sys2 be two versions of Sys where the terms M_k are arbitrary numerals, for $k \in 1..m$. Then $Sys1 \simeq Sys2$.*

7 Conclusions

Perhaps in part because of advances in programming languages, the idea of static checking of security properties seems to be reviving. The Java bytecode verifier is a recent static checker with security objectives [LY96]. In the last couple of years, more sophisticated security checks have been based on self-certification and on information-flow techniques (see for example [Nec97,VSI96]).

This work can be seen as part of that revival. It develops a method for static checking of secrecy properties of programs written in a minimal but expressive programming language, the spi calculus. These programs can be concurrent, and can use cryptography. The method is embodied in a set of typing rules.

The principles and rules developed in this paper are neither necessary nor sufficient for security. They are not necessary because, like most practical static typechecking disciplines, ours is incomplete. They are not sufficient because they ignore all security issues other than secrecy, and because they do not account for how to implement the spi calculus while preserving secrecy properties. However, these principles and rules provide some useful guidelines. Furthermore, the rules are tractable and precise; so we have been able to study them in detail and to prove secrecy properties, establishing the correctness of the informal principles within a formal model.

Acknowledgments

Butler Lampson suggested studying authentication protocols through classification techniques, several years ago. That suggestion was the starting point for this work.

This work took place in the context of collaboration with Andrew Gordon on the spi calculus, so the themes and techniques of this paper owe much to him; Andrew Gordon also commented on a draft of this paper.

Conversations with Mike Burrows, Steve Kent, and Ted Wobber were helpful during the writing of this paper.

References

[AG97a] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[AG97b] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. Technical Report 414, University of Cambridge Computer Laboratory, January 1997. Extended version of both [AG97a] and [AG97c].

[AG97c] M. Abadi and A. D. Gordon. Reasoning about cryptographic protocols in the spi calculus. To appear in the Proceedings of CONCUR'97, 1997.

[AN96] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[BAN89] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, February 1989.

[BN95] M. Boreale and R. De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120(2):279–303, August 1995.

[BR95] M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 1995.

[Den82] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, Mass., 1982.

[DES77] Data encryption standard. Fed. Inform. Processing Standards Pub. 46, National Bureau of Standards, Washington DC, January 1977.

[DH84] R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Gas88] M. Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Company Inc., New York, 1988.

[LY96] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1996.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, October 1991. Appeared in *Logic and Algebra of Specification*, F. L. Bauer, W. Brauer, and H. Schwichtenberg, eds., Springer-Verlag, 1993.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, pages 1–40 and 41–77, September 1992.

[Nec97] G. Necula. Proof-carrying code. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages*, pages 106–119, 1997.

[NS78] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[PS96] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, October 1996.

[Sch96] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., second edition, 1996.

[VSI96] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):1–21, 1996.