A Calculus for Cryptographic Protocols
The Spi Calculus

Martín Abadi
Digital Equipment Corporation
Systems Research Center
ma@pa.dec.com

Andrew D. Gordon
University of Cambridge
Computer Laboratory
adg@cl.cam.ac.uk

Abstract

We introduce the spi calculus, an extension of the pi calculus designed for the description and analysis of cryptographic protocols. We show how to use the spi calculus, particularly for studying authentication protocols. The pi calculus (without extension) suffices for some abstract protocols; the spi calculus enables us to consider cryptographic issues in more detail. We represent protocols as processes in the spi calculus and state their security properties in terms of coarse-grained notions of protocol equivalence.

1 Security and the Pi Calculus

The spi calculus is an extension of the pi calculus [MPW92] with cryptographic primitives. It is designed for the description and analysis of security protocols, such as those for authentication and for electronic commerce. These protocols rely on cryptography and on communication channels with properties like authenticity and privacy. Accordingly, cryptographic operations and communication through channels are the main ingredients of the spi calculus.

We use the pi calculus (without extension) for describing protocols at an abstract level. The pi calculus primitives for channels are simple but powerful. Channels can be created and passed, for example from authentication servers to clients. The scoping rules of the pi calculus guarantee that the environment of a protocol (the attacker) cannot access a channel that it is not explicitly given; scoping is thus the basis of security. In sum, the pi calculus appears as a fairly convenient calculus of protocols for secure communication.

However, the pi calculus does not express the cryptographic operations that are commonly used for implementing channels in distributed systems: it does not include any constructs for encryption and decryption, and these do not seem easy to represent. Since the use of cryptography is notoriously error-prone, we prefer not to abstract it away. We define the spi calculus in order to permit an explicit representation of the use of cryptography in protocols.

There are by now many other notations for describing security protocols. Some, which have long been used in the authentication literature, have a fairly clear connection to the intended implementations of those protocols (see, e.g., [NS78, Lie93]). Their main shortcoming is that they do not provide a precise and solid basis for reasoning about protocols. Other notations (e.g., [BAN89]) are more formal, but their relation to implementations may be more tenuous or subtle. The spi calculus is a middle ground: it is directly executable and it has a precise semantics.

Because the semantics of the spi calculus is not only precise but intelligible, the spi calculus provides a setting for analysing protocols. Specifically, we can express security guarantees as equivalences between spi calculus processes. For example, we can say that a protocol keeps secret a piece of data X by stating that the protocol with X is equivalent to the protocol with X', for any X'. Here, equivalence means equivalence in the eyes of an arbitrary environment. The environment can interact with the protocol, perhaps attempting to create confusion between different messages or sessions. This definition of equivalence yields the desired properties for our security applications. Moreover, in our experience, equivalence is not too hard to prove.

Although the definition of equivalence makes reference to the environment, we do not need to give a model of the environment explicitly. This is one of the main advantages of our approach. Writing such a model can be tedious and can lead to new arbitrariness and error. In particular, it is always difficult to express that the environment can invent random numbers but is not lucky enough to guess the random secrets on which a protocol depends. We resolve this conflict by letting the environment be an arbitrary spi calculus process.

Our approach has some similarities with other recent approaches for reasoning about protocols. Like work based on temporal logics or process algebras (e.g., [GM95, Low96, Sch96]), our method builds on a standard concurrency formalism; this has obvious advantages but it also implies that our method is less intuitive than some based on ad hoc formalisms (e.g., [BAN89]). As in some modal logics (e.g., [ABLP93, LABW92]), we emphasise reasoning about channels and their utterances. As in state-transition models (e.g., [DY81, MCF87, Mil95, Kem89, Mea92]), we are interested in characterising the knowledge of an environment. The unique features of our approach are its reliance on the powerful scoping constructs of the pi calculus; the radical definition of the environment as an arbitrary spi calculus process; and the representation of security properties, both integrity and secrecy, as equivalences.

Our model of protocols is simpler, but poorer, than some models developed for informal mathematical arguments because the spi calculus does not include any notion of probability or complexity (cf. [BR95]). It would be interesting to bridge the gap between the spi calculus and those models, perhaps by giving a probabilistic interpretation for our results.

Contents of this Paper

Section 2 introduces the pi calculus and our method of specifying security properties as equations. Section 3 extends the pi calculus with primitives for shared-key cryptography. Sections 4 defines the formal semantics of the spi calculus. Section 5 discusses how to add primitives for hashing and public-key cryptography to the pi calculus, and Section 6 offers some conclusions. An extended version of this work includes additional material, in particular proof techniques and proofs for examples.

2 Protocols using Restricted Channels

In this section we review the definition of the pi calculus informally. (We give a more formal presentation in Section 4.) We then introduce a new application of the pi calculus, namely its use for the study of security.

2.1 Basics

The pi calculus is a small but extremely expressive programming language. It is an important result of the search for a calculus that could serve as a foundation for concurrent computation, in the same way in which the lambda calculus is a foundation for sequential computation.

Pi calculus programs are systems of independent, parallel processes that synchronise via message-passing handshakes on named channels. The channels a process knows about determine the communication possibilities of the process. Channels may be *restricted*, so that only certain processes may communicate on them. In this respect the pi calculus is similar to earlier process calculi such as CSP [Hoa85] and CCS [Mil89].

What sets the pi calculus apart from earlier calculi is that the scope of a restriction—the program text in which a channel may be used—may change during computation. When a process sends a restricted channel as a message to a process outside the scope of the restriction, the scope is said to *extrude*, that is, it enlarges to embrace the process receiving the channel. Processes in the pi calculus are mobile in the sense that their communication possibilities may change over time; they may learn the names of new channels via scope extrusion. Thus, a channel is a transferable capability for communication.

A central technical idea of this paper is to use the restriction operator and scope extrusion from the pi calculus as a formal model of the possession and communication of secrets, such as cryptographic keys. These features of the pi calculus are essential in our descriptions of security protocols.

2.2 Outline of the Pi Calculus

There are in fact several versions of the pi calculus. Here we review the syntax and semantics of a particular version of the pi calculus. The differences with other versions are mostly orthogonal to our concerns.

We assume an infinite set of *names*, to be used for communication channels, and an infinite set of *variables*. We let m, n, p, q, and r range over names, and let x, y, and z range over variables. The set of *terms* is defined by the grammar:

$L, M, N ::=$	terms
n	name
(M, N)	pair
0	zero
$suc(M)$	successor
x	variable

In the standard pi calculus, names are the only terms. For convenience we have added constructs for pairing and numbers, (M, N), 0, and $suc(M)$, and have also distinguished variables from names.

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

$P, Q, R ::=$	processes
$\overline{M}\langle N\rangle.P$	output
$M(x).P$	input
$P \mid Q$	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

In $(\nu n)P$, the name n is bound in P. In $M(x).P$, the variable x is bound in P. In $let\ (x, y) = M\ in\ P$, the variables x and y are bound in P. In $case\ M\ of\ 0 : P\ suc(x) : Q$, the variable x is bound in the second branch, Q. We write $P[M/x]$ for the outcome of replacing each free occurrence of x in process P with the term M, and identify processes up to renaming of bound variables and names. We adopt the abbreviation $\overline{M}\langle N\rangle$ for $\overline{M}\langle N\rangle.\mathbf{0}$.

Intuitively, the constructs of the pi calculus have the following meanings:

- The basic computation and synchronisation mechanism in the pi calculus is *interaction*, in which a term N is communicated from an output process to an input process via a named channel, m.
  - An *output process* $\overline{m}\langle N\rangle.P$ is ready to output on channel m. If an interaction occurs, term N is communicated on m and then process P runs.
  - An *input process* $m(x).P$ is ready to input from channel m. If an interaction occurs in which N is communicated on m, then process $P[N/x]$ runs.

  (The general forms $\overline{M}\langle N\rangle.P$ and $M(x).P$ of output and input allow for the channel to be an arbitrary term M. The only useful cases are for M to be a name, or a variable that gets instantiated to a name.)
- A *composition* $P \mid Q$ behaves as processes P and Q running in parallel. Each may interact with the other on channels known to both, or with the outside world, independently of the other.
- A *restriction* $(\nu n)P$ is a process that makes a new, private name n, which may occur in P, and then behaves as P.

- A *replication* $!P$ behaves as an infinite number of copies of P running in parallel.
- A *match* $[M \mathit{ is } N]P$ behaves as P provided that terms M and N are the same; otherwise it is stuck, that is, it does nothing.
- The *nil* process $\mathbf{0}$ does nothing.

Since we added pairs and integers, we have two new process forms:

- A *pair splitting* process $\mathit{let}\ (x, y) = M\ \mathit{in}\ P$ behaves as $P[N/x][L/y]$ if term M is the pair (N, L), and otherwise it is stuck.
- An *integer case* process $\mathit{case}\ M\ \mathit{of}\ 0 : P\ \mathit{suc}(x) : Q$ behaves as P if term M is 0, as $Q[N/x]$ if M is $\mathit{suc}(N)$, and otherwise is stuck.

We write $P \simeq Q$ to mean that the behaviours of the processes P and Q are indistinguishable. In other words, a third process R cannot distinguish running in parallel with P from running in parallel with Q; as far as R can tell, P and Q have the same properties (more precisely, the same safety properties). We define the relation $\simeq$ in Section 4.2 as a form of testing equivalence. For now, it suffices to understand $\simeq$ informally.

2.3 Examples using Restricted Channels

Next we show how to express some abstract security protocols in the pi calculus. In security protocols, it is common to find channels on which only a given set of principals is allowed to send data or listen. The set of principals may expand in the course of a protocol run, for example as the result of channel establishment. Remarkably, it is easy to model this property of channels in the pi calculus, via the restriction operation; the expansion of the set of principals that can access a channel corresponds to scope extrusion.

2.3.1 A first example

Our first example is extremely basic. In this example, there are two principals A and B that share a channel, c_{AB}; only A and B can send data or listen on this channel. The protocol is simply that A uses c_{AB} for sending a single message M to B. In informal notation, we may write this protocol as follows:

$$\text{Message 1} \quad A \to B : \quad M \quad \text{on } c_{AB}$$

A first pi calculus description of this protocol is:

$$\begin{aligned} A(M) &\triangleq \overline{c_{AB}}\langle M\rangle \\ B &\triangleq c_{AB}(x).\mathbf{0} \\ \mathit{Inst}(M) &\triangleq (\nu c_{AB})(A(M) \mid B) \end{aligned}$$

The processes $A(M)$ and B describe the two principals, and $\mathit{Inst}(M)$ describes (one instance of) the whole protocol. The channel c_{AB} is restricted; intuitively, this achieves the effect that only A and B have access to c_{AB}.

In these definitions, $A(M)$ and $\mathit{Inst}(M)$ are processes parameterised by M. More formally, we view A and Inst as functions that map terms to processes, called abstractions, and treat the M's on the left of $\triangleq$ as bound parameters. Abstractions can of course be instantiated (applied); for example, the instantiation $A(0)$ yields $\overline{c_{AB}}\langle 0\rangle$. The standard rules of substitution govern application, forbidding parameter captures; for example, expanding $\mathit{Inst}(c_{AB})$ would require a renaming of the bound occurrence of c_{AB} in the definition of Inst.

The first pi calculus description of the protocol may seem a little futile because, according to it, B does nothing with its input. A more useful and general description says that B runs a process F with its input. We revise our definitions as follows:

$$\begin{aligned} A(M) &\triangleq \overline{c_{AB}}\langle M\rangle \\ B &\triangleq c_{AB}(x).F(x) \\ \mathit{Inst}(M) &\triangleq (\nu c_{AB})(A(M) \mid B) \end{aligned}$$

Informally, $F(x)$ is simply the result of applying F to x. More formally, F is an abstraction, and $F(x)$ is an instantiation of the abstraction. We adopt the convention that the bound parameters of the protocol (in this case, M, c_{AB}, and x) cannot occur free in F.

This protocol has two important properties:

- Authenticity (or integrity): B always applies F to the message M that A sends; an attacker cannot cause B to apply F to some other message.
- Secrecy: The message M cannot be read in transit from A to B: if F does not reveal M, then the whole protocol does not reveal M.

The secrecy property can be stated in terms of equivalences: if $F(M) \simeq F(M')$, for any M, M', then $\mathit{Inst}(M) \simeq \mathit{Inst}(M')$. This means that if $F(M)$ is indistinguishable from $F(M')$, then the protocol with message M is indistinguishable from the protocol with message M'.

There are many sensible ways of formalising the authenticity property. In particular, it may be possible to use notions of refinement or a suitable program logic. However, we choose to write authenticity as an equivalence, for economy. This equivalence compares the protocol with another protocol. Our intent is that the latter protocol serves as a specification. In this case, the specification is:

$$\begin{aligned} A(M) &\triangleq \overline{c_{AB}}\langle M\rangle \\ B_{spec}(M) &\triangleq c_{AB}(x).F(M) \\ \mathit{Inst}_{spec}(M) &\triangleq (\nu c_{AB})(A(M) \mid B_{spec}(M)) \end{aligned}$$

The principal A is as usual, but the principal B is replaced with a variant $B_{spec}(M)$; this variant receives an input from A and then acts like B when B receives M. We may say that $B_{spec}(M)$ is a "magical" version of B that knows the message M sent by A, and similarly Inst_{spec} is a "magical" version of Inst.

Although the specification and the protocol are similar in structure, the specification is more evidently "correct" than the protocol. Therefore, we take the following equivalence as our authenticity property: $\mathit{Inst}(M) \simeq \mathit{Inst}_{spec}(M)$, for any M.

In summary, we have:

Authenticity: $\mathit{Inst}(M) \simeq \mathit{Inst}_{spec}(M)$, for any M.

Secrecy: $\mathit{Inst}(M) \simeq \mathit{Inst}(M')$ if $F(M) \simeq F(M')$, for any M, M'.

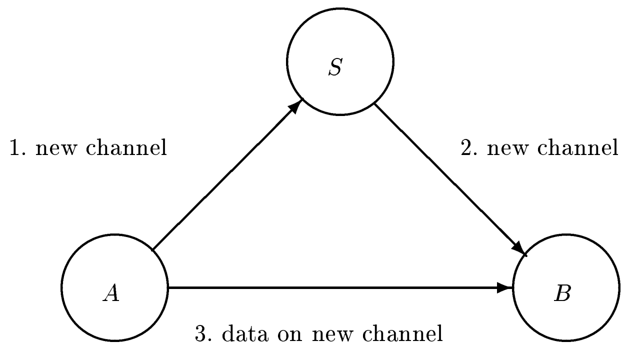

Figure 1: Structure of the Wide Mouthed Frog Protocol

Each of these equivalences means that two processes being equated are indistinguishable, even when an active attacker is their environment. Neither of these equivalences would hold without the restriction of channel c_{AB}.

2.3.2 An example with channel establishment

A more interesting variant of our first example is obtained by adding a channel establishment phase. In this phase, before communication of data, the principals A and B obtain a new channel with the help of a server S.

There are many different ways of establishing a channel, even at the abstract level at which we work here. The one we describe is inspired by the Wide Mouthed Frog protocol [BAN89], which has the basic structure shown in Figure 1.

We consider an abstract and simplified version of the Wide Mouthed Frog protocol. Our version is abstract in that we deal with channels instead of keys; it is simplified in that channel establishment and data communication happen only once (so there is no need for timestamps). In the next section we show how to treat keys and how to allow many instances of the protocol, with an arbitrary number of messages.

Informally, our version is:

Message 1	$A \to S:$	c_{AB}	on c_{AS}
Message 2	$S \to B:$	c_{AB}	on c_{SB}
Message 3	$A \to B:$	M	on c_{AB}

Here c_{AS} is a channel that A and S share initially, c_{SB} is a channel that S and B share initially, and c_{AB} is a channel that A creates for communication with B. After passing the channel c_{AB} to B through S, A sends a message M on c_{AB}. Note that S does not use the channel, but only transmits it.

In the pi calculus, we formulate this protocol as follows:

$$\begin{aligned}
A(M) &\triangleq (\nu c_{AB})\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle M\rangle \\
S &\triangleq c_{AS}(x).\overline{c_{SB}}\langle x\rangle \\
B &\triangleq c_{SB}(x).x(y).F(y) \\
\mathit{Inst}(M) &\triangleq (\nu c_{AS})(\nu c_{SB})(A(M) \mid S \mid B)
\end{aligned}$$

Here we write $F(y)$ to represent what B does with the message y that it receives, as in the previous example. The restrictions on the channels c_{AS}, c_{SB}, and c_{AB} reflect the expected privacy guarantees for these channels. The most salient new feature of this specification is the use of scope extrusion: A generates a fresh channel c_{AB}, and then sends it out of scope to B via S. We could not have written this description in formalisms such as CCS or CSP; the use of the pi calculus is important.

For discussing authenticity, we introduce the following specification:

$$\begin{aligned}
A(M) &\triangleq (\nu c_{AB})\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle M\rangle \\
S &\triangleq c_{AS}(x).\overline{c_{SB}}\langle x\rangle \\
B_{spec}(M) &\triangleq c_{SB}(x).x(y).F(M) \\
\mathit{Inst}_{spec}(M) &\triangleq (\nu c_{AS})(\nu c_{SB})(A(M) \mid S \mid B_{spec}(M))
\end{aligned}$$

According to this specification, the message M is communicated "magically": the process F is applied to the message M that A sends independently of whatever happens during the rest of the protocol run.

We obtain the following authenticity and secrecy properties:

Authenticity: $\mathit{Inst}(M) \simeq \mathit{Inst}_{spec}(M)$, for any M.

Secrecy: $\mathit{Inst}(M) \simeq \mathit{Inst}(M')$ if $F(M) \simeq F(M')$, for any M, M'.

Again, these properties hold because of the scoping rules of the pi calculus.

3 Protocols using Cryptography

Just as there are several versions of the pi calculus, there are several versions of the spi calculus. These differ in particular in what cryptographic constructs they include. In this section we introduce a relatively simple spi calculus, namely the pi calculus extended with primitives for shared-key cryptography. We then write several protocols that use shared-key cryptography in this calculus.

Throughout the paper, we often refer to the calculus presented in this section as "the" spi calculus; but we define other versions of the spi calculus in Section 5.

3.1 The Spi Calculus with Shared-Key Cryptography

The syntax of the spi calculus is an extension of that of the pi calculus. In order to represent encrypted messages, we add a clause to the syntax of terms:

$L, M, N ::=$	terms	
$\quad \ldots$		as in Section 2.2
$\quad \{M\}_N$		shared-key encryption

In order to represent decryption, we add a clause to the syntax of processes:

$P, Q ::=$	processes	
$\quad \ldots$		as in Section 2.2
$\quad$ *case* L *of* $\{x\}_N$ *in* P		shared-key decryption

The variable x is bound in P.

Intuitively, the meaning of the new constructs is as follows:

- The term $\{M\}_N$ represents the ciphertext obtained by encrypting the term M under the key N using a shared-key cryptosystem such as DES [DES77].
- The process *case* L *of* $\{x\}_N$ *in* P attempts to decrypt the term L with the key N. If L is a ciphertext of the form $\{M\}_N$, then the process behaves as $P[M/x]$. Otherwise the process is stuck.

Implicit in this definition are some standard but significant assumptions about cryptography:

- The only way to decrypt an encrypted packet is to know the corresponding key.
- An encrypted packet does not reveal the key that was used to encrypt it.
- There is sufficient redundancy in messages so that the decryption algorithm can detect whether a ciphertext was encrypted with the expected key.

It is not assumed that all messages contain information that allows each principal to recognise its own messages (cf. [BAN89]).

The semantics of the spi calculus can be formalised in much the same way as the semantics of the pi calculus. We carry out this formalisation in Section 4. Again, we write $P \simeq Q$ to mean that the behaviours of the processes P and Q are indistinguishable. The notion of indistinguishability is complicated by the presence of cryptography. As an example of these complications, consider the following process:

$$P(M) \triangleq (\nu K)\overline{c}\langle\{M\}_K\rangle$$

This process simply sends M under a new key K on a public channel c; the key K is not transmitted. Intuitively, we would like to equate $P(M)$ and $P(M')$, for any M and M', because an observer cannot discover K and hence cannot tell whether M or M' is sent under K. On the other hand, $P(M)$ and $P(M')$ are clearly different, since they transmit different messages on c. Our equivalence $\simeq$ is coarse-grained enough to equate $P(M)$ and $P(M')$.

3.2 Examples using Shared-Key Cryptography

The spi calculus enables more detailed descriptions of security protocols than the pi calculus. While the pi calculus enables the representation of channels, the spi calculus also enables the representation of the channel implementations in terms of cryptography. In this section we show a few example cryptographic protocols.

As in the pi calculus, scoping is the basis of security in the spi calculus. In particular, restriction can be used to model the creation of fresh, unguessable cryptographic keys. Restriction can also be used to model the creation of fresh nonces of the sort used in challenge-response exchanges.

Security properties can still be expressed as equivalences, although the notion of equivalence is more delicate, as we have discussed.

3.2.1 A first cryptographic example

Our first example is a cryptographic version of the example of Section 2.3.1. We consider two principals A and B that share a key K_{AB}; in addition, we assume there is a public channel c_{AB} that A and B can use for communication, but which is in no way secure. The protocol is simply that A sends a message M under K_{AB} to B, on c_{AB}.

Informally, we write this protocol as follows:

$$\text{Message 1} \quad A \to B: \quad \{M\}_{K_{AB}} \quad \text{on } c_{AB}$$

In the spi calculus, we write:

$$\begin{aligned} A(M) &\triangleq \overline{c_{AB}}\langle\{M\}_{K_{AB}}\rangle \\ B &\triangleq c_{AB}(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AB}}\ \mathit{in}\ F(y) \\ \mathit{Inst}(M) &\triangleq (\nu K_{AB})(A(M) \mid B) \end{aligned}$$

According to this definition, A sends $\{M\}_{K_{AB}}$ on c_{AB} while B listens for a message on c_{AB}. Given such a message, B attempts to decrypt it using K_{AB}; if this decryption succeeds, B applies F to the result. The assumption that A and B share K_{AB} gives rise to the restriction on K_{AB}, which is syntactically legal and meaningful although K_{AB} is not used as a channel. On the other hand, c_{AB} is not restricted, since it is a public channel. Other principals may send messages on c_{AB}, so B may attempt to decrypt a message not encrypted under K_{AB}; in that case, the protocol will get stuck. We are not concerned about this possibility, but it would be easy enough to avoid it by writing a slightly more elaborate program for B.

We use the following specification:

$$\begin{aligned} A(M) &\triangleq \overline{c_{AB}}\langle\{M\}_{K_{AB}}\rangle \\ B_{spec}(M) &\triangleq c_{AB}(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AB}}\ \mathit{in}\ F(M) \\ \mathit{Inst}_{spec}(M) &\triangleq (\nu K_{AB})(A(M) \mid B_{spec}(M)) \end{aligned}$$

and we obtain the properties:

Authenticity: $\mathit{Inst}(M) \simeq \mathit{Inst}_{spec}(M)$, for any M.

Secrecy: $\mathit{Inst}(M) \simeq \mathit{Inst}(M')$ if $F(M) \simeq F(M')$, for any M, M'.

Intuitively, authenticity holds even if the key K_{AB} is somehow compromised after its use. Many factors can contribute to key compromise, for example incompetence on the part of protocol participants, and malice and brute force on the part of attackers. We cannot model all these factors, but we can model deliberate key publication, which is in a sense the most extreme of them. It suffices to make a small change in the definitions of B and B_{spec}, so that they send K_{AB} on a public channel after receiving $\{M\}_{K_{AB}}$. This change preserves the authenticity equation, but clearly not the secrecy equation.

3.2.2 An example with key establishment

In cryptographic protocols, the establishment of new channels often means the exchange of new keys. There are many methods (most of them flawed) for key exchange. The following example is the cryptographic version of that of Section 2.3.2; it uses a simplified (one-shot) form of the Wide Mouthed Frog key exchange.

In the Wide Mouthed Frog protocol, the principals A and B share keys K_{AS} and K_{SB} respectively with a server S. When A and B want to communicate securely, A creates a new key K_{AB}, sends it to the server under K_{AS}, and the server forwards it to B under K_{SB}. All communication being protected by encryption, it can happen through public channels, which we write c_{AS}, c_{SB}, and c_{AB}. Informally, a simplified version of this protocol is:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow S: & \{K_{AB}\}_{K_{AS}} \quad \text{on } c_{AS} \\ \text{Message 2} & S \rightarrow B: & \{K_{AB}\}_{K_{SB}} \quad \text{on } c_{SB} \\ \text{Message 3} & A \rightarrow B: & \{M\}_{K_{AB}} \quad\; \text{on } c_{AB} \end{array}$$

In the spi calculus, we can express this message sequence as follows:

$$\begin{array}{rcl} A(M) & \triangleq & (\nu K_{AB})(\overline{c_{AS}}\langle\{K_{AB}\}_{K_{AS}}\rangle.\overline{c_{AB}}\langle\{M\}_{K_{AB}}\rangle) \\ S & \triangleq & c_{AS}(x).case\ x\ of\ \{y\}_{K_{AS}}\ in\ \overline{c_{SB}}\langle\{y\}_{K_{SB}}\rangle \\ B & \triangleq & c_{SB}(x).case\ x\ of\ \{y\}_{K_{SB}}\ in \\ & & c_{AB}(z).case\ z\ of\ \{w\}_y\ in\ F(w) \\ Inst(M) & \triangleq & (\nu K_{AS})(\nu K_{SB})(A(M) \mid S \mid B) \end{array}$$

where $F(w)$ is a process representing the rest of the behaviour of B upon receiving a message w. Notice the essential use of scope extrusion: A generates the key K_{AB} and sends it out of scope to B via S.

In the usual pattern, we introduce a specification for discussing authenticity:

$$\begin{array}{rcl} A(M) & \triangleq & (\nu K_{AB})(\overline{c_{AS}}\langle\{K_{AB}\}_{K_{AS}}\rangle.\overline{c_{AB}}\langle\{M\}_{K_{AB}}\rangle) \\ S & \triangleq & c_{AS}(x).case\ x\ of\ \{y\}_{K_{AS}}\ in\ \overline{c_{SB}}\langle\{y\}_{K_{SB}}\rangle \\ B_{spec}(M) & \triangleq & c_{SB}(x).case\ x\ of\ \{y\}_{K_{SB}}\ in \\ & & c_{AB}(z).case\ z\ of\ \{w\}_y\ in\ F(M) \\ Inst_{spec}(M) & \triangleq & (\nu K_{AS})(\nu K_{SB})(A(M) \mid S \mid B_{spec}(M)) \end{array}$$

One may be concerned about the apparent complexity of this specification. On the other hand, despite its complexity, the specification is still more evidently "correct" than the protocol. In particular, it is still evident that $B_{spec}(M)$ applies F to the data M from A, rather than to some other message chosen as the result of error or attack.

We obtain the usual properties of authenticity and secrecy:

Authenticity: $Inst(M) \simeq Inst_{spec}(M)$, for any M.

Secrecy: $Inst(M) \simeq Inst(M')$ if $F(M) \simeq F(M')$, for any M, M'.

3.2.3 A complete authentication example (with a flaw)

In the examples discussed so far, channel establishment and data communication happen only once. As we demonstrate now, it is a simple matter of programming to remove this restriction and to represent more sophisticated examples with many sessions between many principals. However, as the intricacy of our examples increases, so does the opportunity for error. This should not be construed as a limitation of our approach, but rather as the sign of an intrinsic difficulty: many of the mistakes in authentication protocols arise from confusion between sessions.

We consider a system with a server S and n other principals. We use the terms $suc(0)$, $suc(suc(0))$, $\ldots$, which we abbreviate to $\underline{1}$, $\underline{2}$, $\ldots$, as the names of these other principals. We assume that each principal has an input channel; these input channels are public and have the names c_1, c_2, $\ldots$, c_n and c_S. We also assume that the server shares a pair of keys with each other principal, one key for each direction: principal i uses key K_{iS} to send to S and key K_{Si} to receive from S, for $1 \leq i \leq n$.

We extend our standard example to this system of $n+1$ principals, with the following message sequence:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow S: & A, \{B, K_{AB}\}_{K_{AS}} \quad \text{on } c_S \\ \text{Message 2} & S \rightarrow B: & \{A, K_{AB}\}_{K_{SB}} \quad\;\, \text{on } c_B \\ \text{Message 3} & A \rightarrow B: & A, \{M\}_{K_{AB}} \qquad\;\; \text{on } c_B \end{array}$$

Here A and B range over the n principals. The names A and B appear in messages in order to avoid ambiguity; when these names appear in clear, they function as hints that help the recipient choose the appropriate key for decryption of the rest of the message. The intent is that the protocol can be used by any pair of principals, arbitrarily often; concurrent runs are allowed. As it stands, the protocol has obvious flaws; we discuss it in order to explain our method for representing it in the spi calculus.

In our spi calculus representation, we use several convenient abbreviations. Firstly, we rely on pair splitting on input and on decryption:

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

where variable y is fresh. Secondly, we need the standard notation for the composition of a finite set of processes. Given a finite family of processes $P_1, \ldots, P_k$, we let $\prod_{i \in 1..k} P_i$ be their k-way composition $P_1 \mid \cdots \mid P_k$. Finally, we omit the inner parentheses from an encrypted pair of the form $\{(N, N')\}_{N''}$, and simply write $\{N, N'\}_{N''}$, as is common in informal descriptions.

Informally, an instance of the protocol is determined by a choice of parties (who is A and who is B) and by the message sent after key establishment. More formally, an instance I is a triple (i, j, M) such that i and j are principals and M is a message. We say that i is the source address and j the destination address of the instance. Moreover, we assume that there is an abstraction F representing the behaviour of any principal after receipt of Message 3 of the protocol. For an instance (i, j, M) that runs as intended, the argument to F is the triple $(\underline{i}, \underline{j}, M)$.

Given an instance (i, j, M), the following process corresponds to the role of A:

$$Send(i,j,M) \triangleq (\nu K)(\overline{c_S}\langle(\underline{i}, \{\underline{j}, K\}_{K_{iS}})\rangle \mid \overline{c_j}\langle(\underline{i}, \{M\}_K)\rangle)$$

The sending process creates a key K and sends it to the server, along with the names $\underline{i}$ and $\underline{j}$ of the principals of the instance. The sending process also sends M under K, along with its name $\underline{i}$. We have put the two messages in parallel, somewhat arbitrarily; putting them in sequence would have much the same effect.

The following process corresponds to the role of B for principal j:

$$\begin{aligned} Recv(j) \triangleq\ & c_j(y_{cipher}).case\ y_{cipher}\ of\ \{x_A, x_{key}\}_{K_{Sj}}\ in \\ & c_j(z_A, z_{cipher}).[x_A\ is\ z_A] \\ & case\ z_{cipher}\ of\ \{z_{plain}\}_{x_{key}}\ in\ F(x_A, \underline{j}, z_{plain}) \end{aligned}$$

The receiving process waits for a message y_{cipher} from the server, extracts a key x_{key} from this message, then waits for a message z_{cipher} under this key, and finally applies F to the name x_A of the presumed sender, to its own name $\underline{j}$, and to the contents z_{plain} of the message. The variables x_A and z_A are both intended as the name of the sending process, so they are expected to match.

The server S is the same for all instances:

$$\begin{aligned} S \triangleq\ & c_S(x_A, x_{cipher}). \\ & \textstyle\prod_{i\in 1..n}[x_A\ is\ \underline{i}]\ case\ x_{cipher}\ of\ \{x_B, x_{key}\}_{K_{iS}}\ in \\ & \textstyle\prod_{j\in 1..n}[x_B\ is\ \underline{j}]\ \overline{c_j}\langle\{x_A, x_{key}\}_{K_{Sj}}\rangle \end{aligned}$$

The variable x_A is intended as the name of the sending process, x_B as the name of the receiving process, x_{key} as the new key, and x_{cipher} as the encrypted part of the first message of the protocol. In the code for the server, we program an n-way branch on the name x_A by using a parallel composition of processes indexed by $i \in 1..n$. We also program an n-way branch on the name x_B, similarly. (This casual use of multiple threads is characteristic of the pi calculus; in practice the branch could be implemented more efficiently, but here we are interested only in the behaviour of the server, not in its efficient implementation.)

Finally we define a whole system, parameterised on a list of instances:

$$\begin{aligned} Sys(I_1, \ldots, I_m) \triangleq\ & (\nu \vec{K_{iS}})(\nu \vec{K_{Sj}}) \\ & (Send(I_1) \mid \cdots \mid Send(I_m) \mid \\ & !S \mid \\ & !Recv(1) \mid \cdots \mid !Recv(n)) \end{aligned}$$

where $(\nu \vec{K_{iS}})(\nu \vec{K_{Sj}})$ stands for:

$$(\nu K_{1S}) \ldots (\nu K_{nS})(\nu K_{S1}) \ldots (\nu K_{Sn})$$

The expression $Sys(I_1, \ldots, I_m)$ represents a system with m instances of the protocol. The server is replicated; in addition, the replication of the receiving processes means that each principal is willing to play the role of receiver in any number of runs of the protocol in parallel. Thus, any two runs of the protocol can be simultaneous, even if they involve the same principals.

As before, we write a specification by modifying the protocol. For this specification, we revise the sending and the receiving processes, but not the server:

$$\begin{aligned} Send_{spec}(i,j,M) \triangleq\ & (\nu p)(Send(i,j,p) \mid p(x).F(\underline{i}, \underline{j}, M)) \\ Recv_{spec}(j) \triangleq\ & c_j(y_{cipher}). \\ & case\ y_{cipher}\ of\ \{x_A, x_{key}\}_{K_{Sj}}\ in \\ & c_j(z_A, z_{cipher}).[x_A\ is\ z_A] \\ & case\ z_{cipher}\ of\ \{z_{plain}\}_{x_{key}}\ in \\ & \overline{z_{plain}}\langle * \rangle \\ Sys_{spec}(I_1, \ldots, I_m) \triangleq\ & (\nu \vec{K_{iS}})(\nu \vec{K_{Sj}}) \\ & (Send_{spec}(I_1) \mid \cdots \mid Send_{spec}(I_m) \mid \\ & !S \mid \\ & !Recv_{spec}(1) \mid \cdots \mid !Recv_{spec}(n)) \end{aligned}$$

In this specification, the sending process for instance (i, j, M) is as in the implementation, except that it sends a fresh channel name p instead of M, and runs $F(\underline{i}, \underline{j}, M)$ when it receives any message on p. The receiving process in the specification is identical to that in the implementation, except that $F(y_A, \underline{j}, z_{plain})$ is replaced with $\overline{z_{plain}}\langle * \rangle$, where the symbol $*$ represents a fixed but arbitrary message. The variable z_{plain} will be bound to the fresh name p for the corresponding instance of the protocol. Thus, the receiving process will signal on p, triggering the execution of the appropriate process $F(\underline{i}, \underline{j}, M)$.

A crucial property of this specification is that the only occurrences of F are bundled into the description of the sending process. There, F is applied to the desired parameters, $(\underline{i}, \underline{j}, M)$. Hence it is obvious that an instance (i, j, M) will cause the execution of $F(\underline{i}', \underline{j}', M')$ only if i' is i, j' is j, and M' is M. Therefore, despite its complexity, the specification is more obviously "correct" than the implementation.

Much as in previous examples, we would like the protocol to have the following authenticity property:

$$Sys(I_1, \ldots, I_m) \simeq Sys_{spec}(I_1, \ldots, I_m), \\ \text{for any instances } I_1, \ldots, I_m.$$

Unfortunately, the protocol is vulnerable to a replay attack that invalidates the authenticity equation. Consider the system $Sys(I, I')$ where $I = (i, j, M)$ and $I' = (i, j, M')$. An attacker can replay messages of one instance and get them mistaken for messages of the other instance, causing M to be passed twice to F. Thus, $Sys(I, I')$ can be made to execute two copies of $F(\underline{i}, \underline{j}, M)$. In contrast, no matter what an attacker does, $Sys_{spec}(I, I')$ will run each of $F(\underline{i}, \underline{j}, M)$ and $F(\underline{i}, \underline{j}, M')$ at most once. The authenticity equation therefore does not hold. (We can disprove it formally by defining an attacker that distinguishes $Sys(I, I')$ and $Sys_{spec}(I, I')$, within the spi calculus.)

3.2.4 A complete authentication example (repaired)

Now we improve the protocol of the previous section by adding nonce handshakes as protection against replay attacks. The Wide Mouthed Frog protocol uses timestamps instead of handshakes. The treatment of timestamps in

$$
\begin{array}{rcl}
\mathit{Send}(i,j,M) & \triangleq & \overline{c_S}\langle \underline{i} \rangle \mid \\
& & c_i(x_{nonce}).(\nu K)(\overline{c_S}\langle(\underline{i}, \{\underline{i}, \underline{i}, \underline{j}, K, x_{nonce}\}_{K_{iS}})\rangle \mid \overline{c_j}\langle(\underline{i}, \{M\}_K)\rangle) \\
S & \triangleq & c_S(x_A).\prod_{i\in 1..n}[x_A \ \mathit{is}\ \underline{i}]\ (\nu N_S)(\overline{c_i}\langle N_S\rangle \mid \\
& & c_S(x'_A, x_{cipher}).[x'_A \ \mathit{is}\ \underline{i}] \\
& & \mathit{case}\ x_{cipher}\ \mathit{of}\ \{y_A, z_A, x_B, x_{key}, x_{nonce}\}_{K_{iS}}\ \mathit{in} \\
& & \prod_{j\in 1..n}[y_A \ \mathit{is}\ \underline{i}]\ [z_A \ \mathit{is}\ \underline{i}]\ [x_B \ \mathit{is}\ \underline{j}]\ [x_{nonce}\ \mathit{is}\ N_S] \\
& & (\overline{c_j}\langle * \rangle \mid c_S(y_{nonce}).\overline{c_j}\langle\{S, \underline{i}, \underline{j}, x_{key}, y_{nonce}\}_{K_{Sj}}\rangle)) \\
\mathit{Recv}(j) & \triangleq & c_j(w).(\nu N_B)(\overline{c_S}\langle N_B\rangle \mid \\
& & c_j(y_{cipher}). \\
& & \mathit{case}\ y_{cipher}\ \mathit{of}\ \{x_S, x_A, x_B, x_{key}, y_{nonce}\}_{K_{Sj}}\ \mathit{in} \\
& & \prod_{i\in 1..n}[x_S \ \mathit{is}\ S]\ [x_A \ \mathit{is}\ \underline{i}]\ [x_B \ \mathit{is}\ \underline{j}]\ [y_{nonce}\ \mathit{is}\ N_B] \\
& & c_j(z_A, z_{cipher}).[z_A \ \mathit{is}\ x_A] \\
& & \mathit{case}\ z_{cipher}\ \mathit{of}\ \{z_{plain}\}_{x_{key}}\ \mathit{in}\ F(\underline{i}, \underline{j}, z_{plain})) \\
\mathit{Sys}(I_1,\ldots,I_m) & \triangleq & (\nu \vec{K_{iS}})(\nu \vec{K_{Sj}}) \\
& & (\mathit{Send}(I_1) \mid \cdots \mid \mathit{Send}(I_m) \mid \\
& & !S \mid \\
& & !\mathit{Recv}(1) \mid \cdots \mid !\mathit{Recv}(n))
\end{array}
$$

Figure 2: Formalisation of the Seven-Message Protocol

the spi calculus is possible, but it requires additional elements, including at least a rudimentary account of clock synchronisation. Protocols that use handshakes are fundamentally more self-contained than protocols that use timestamps; therefore, handshakes make for clearer examples.

Informally, our new protocol is:

Message 1	$A \to S:$	A	on c_S
Message 2	$S \to A:$	N_S	on c_A
Message 3	$A \to S:$	$A, \{A, A, B, K_{AB}, N_S\}_{K_{AS}}$	on c_S
Message 4	$S \to B:$	$*$	on c_B
Message 5	$B \to S:$	N_B	on c_S
Message 6	$S \to B:$	$\{S, A, B, K_{AB}, N_B\}_{K_{SB}}$	on c_B
Message 7	$A \to B:$	$A, \{M\}_{K_{AB}}$	on c_B

Messages 1 and 2 are the request for a challenge and the challenge, respectively. The challenge is N_S, a nonce created by S; the nonce must not have been used before for this purpose. Obviously the nonce is not secret, but it must be unpredictable (for otherwise an attacker could simulate a challenge and later replay the response [AN96]). In Message 3, A says that A and B can communicate under K_{AB}, sometime after receipt of N_S. All the components A, B, K_{AB}, N_S appear explicitly in the message, for safety [AN96], but A could perhaps be elided. The presence of N_S in Message 3 proves the freshness of the message. In Message 4, $*$ represents a fixed but arbitrary message; S uses $*$ to signal that it is ready for a nonce challenge N_B from B. In Message 6, S says that A says that A and B can communicate under K_{AB}, sometime after receipt of N_B. The first field of the encrypted portions of Messages 3 and 6 (A or S) is included in order to distinguish these messages; it serves as a "direction bit". Finally, Message 7 is the transmission of data under K_{AB}.

The messages of this protocol have many components. For the spi calculus representation it is therefore convenient to generalise our syntax of pairs and pair splitting to arbitrary tuples. We use the following standard abbreviations:

$$(N_1, \ldots, N_{k+1}) \triangleq ((N_1, \ldots, N_k), N_{k+1})$$

$$\begin{array}{rcl} \mathit{let}\ (x_1, \ldots, x_{k+1}) = N\ \mathit{in}\ P & \triangleq & \mathit{let}\ (y, x_{k+1}) = N\ \mathit{in} \\ & & \mathit{let}\ (x_1, \ldots, x_k) = y\ \mathit{in}\ P \end{array}$$

where variable y is fresh.

In the spi calculus, we represent the nonces of this protocol as newly created names. We obtain the spi calculus expressions given in Figure 2. In those expressions, the names N_S and N_B represent the nonces. The variable subscripts are hints that indicate what the corresponding variables should represent; for example, x_A, x'_A, y_A, and z_A are all expected to be the name of the sending process, and x_{nonce} and y_{nonce} are expected to be the nonces generated by S and B, respectively.

The definition of Sys_{spec} is exactly analogous to that of the previous section, so we omit it. We obtain the authenticity property:

$$\mathit{Sys}(I_1, \ldots, I_m) \simeq \mathit{Sys}_{spec}(I_1, \ldots, I_m), \quad \text{for any instances } I_1, \ldots, I_m.$$

This property holds because of the use of nonces. In particular, the replay attack of Section 3.2.3 can no longer distinguish $\mathit{Sys}(I_1, \ldots, I_m)$ and $\mathit{Sys}_{spec}(I_1, \ldots, I_m)$.

As a secrecy property, we would like to express that there is no way for an external observer to tell apart two executions of the system with identical participants but different messages. The secrecy property should therefore assert that the protocol does not reveal any information about the contents of exchanged messages if none is revealed after the key exchange.

In order to express that no information is revealed after the key exchange, we introduce the following definition. We say that a pair of instances (i, j, M) and (i', j', M') is *indistinguishable* if the two instances have the same source and destination addresses ($i = i'$ and $j = j'$) and if $F(\underline{i}, \underline{j}, M) \simeq F(\underline{i}, \underline{j}, M')$.

Our definition of secrecy is that, if each pair (I_1, J_1), $\ldots$, (I_m, J_m) is indistinguishable, then $\mathit{Sys}(I_1, \ldots, I_m) \simeq \mathit{Sys}(J_1, \ldots, J_m)$. This means that an observer cannot distinguish two systems parameterised by two sets of indistinguishable instances. This property holds for our protocol.

In summary, we have:

Authenticity: $Sys(I_1, \ldots, I_m) \simeq Sys_{spec}(I_1, \ldots, I_m)$, for any instances $I_1, \ldots, I_m$.

Secrecy: $Sys(I_1, \ldots, I_m) \simeq Sys(J_1, \ldots, J_m)$, if each pair $(I_1, J_1), \ldots, (I_m, J_m)$ is indistinguishable.

We could ask for a further property of anonymity, namely that the source and the destination addresses of instances be protected from eavesdroppers. However, anonymity holds neither for our protocol nor for most current, practical protocols. It would be easy enough to specify anonymity, should it be relevant.

3.2.5 Discussion

As these examples show, writing a protocol in the spi calculus is essentially analogous to writing it in any programming language with suitable communication and encryption libraries. The main advantage of the spi calculus is its formal precision.

Writing a protocol in the spi calculus may be a little harder than writing it in some of the notations common in the literature. On the other hand, the spi calculus versions are more detailed. They make clear not only what messages are sent but how the messages are generated and how they are checked. These aspects of the spi calculus descriptions add complexity, but they enable finer analysis.

4 Formal Semantics of the Spi Calculus

In this section we give a brief formal treatment of the spi calculus. In Section 4.1 we introduce the reaction relation; $P \to Q$ means there is a reaction amongst the subprocesses of P such that the whole can take a step to process Q. Reaction is the basic notion of computation in both the pi calculus and the spi calculus. In Section 4.2 we give a precise definition of the equivalence relation $\simeq$, which we have used for expressing security properties.

Syntactic Conventions

We write $fn(M)$ and $fn(P)$ for the sets of names free in term M and process P respectively. Similarly, we write $fv(M)$ and $fv(P)$ for the sets of variables free in M and P respectively. We say that a term or process is *closed* to mean that it has no free variables. (To be able to communicate externally, a process must have free names.) The set $Proc = \{P \mid fv(P) = \emptyset\}$ is the set of closed processes.

4.1 The Reaction Relation

The reaction relation is a concise account of computation in the pi calculus introduced by Milner [Mil92], inspired by the Chemical Abstract Machine of Berry and Boudol [BB90]. One thinks of a process as consisting of a chemical solution of molecules waiting to react. A reaction step arises from the interaction of the adjacent molecules $\overline{m}\langle N\rangle.P$ and $m(x).Q$, as follows:

$$\overline{m}\langle N\rangle.P \mid m(x).Q \;\to\; P \mid Q[N/x]$$

Just as one might stir a chemical solution to allow non-adjacent molecules to react, we define a relation, *structural equivalence*, that allows processes to be rearranged so that the rule above is applicable. We first define the *reduction relation* $>$ on closed processes:

$$\begin{array}{rcl}
!P & > & P \mid !P \\
[M \; is \; M]\, P & > & P \\
let\ (x,y) = (M,N)\ in\ P & > & P[M/x][N/y] \\
case\ 0\ of\ 0 : P\ suc(x) : Q & > & P \\
case\ suc(M)\ of\ 0 : P\ suc(x) : Q & > & Q[M/x] \\
case\ \{M\}_N\ of\ \{x\}_N\ in\ P & > & P[M/x]
\end{array}$$

We let structural equivalence, $\equiv$, be the least relation on closed processes that satisfies the following equations and rules:

$$\begin{array}{rcll}
P \mid \mathbf{0} & \equiv & P & \\
P \mid Q & \equiv & Q \mid P & \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \\
(\nu m)(\nu n)P & \equiv & (\nu n)(\nu m)P & \\
(\nu n)\mathbf{0} & \equiv & \mathbf{0} & \\
(\nu n)(P \mid Q) & \equiv & P \mid (\nu n)Q & \quad \text{if } n \notin fn(P)
\end{array}$$

$$\frac{P > Q}{P \equiv Q} \qquad \frac{}{P \equiv P}$$

$$\frac{P \equiv Q}{Q \equiv P} \qquad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R}$$

$$\frac{P \equiv P'}{P \mid Q \equiv P' \mid Q} \qquad \frac{P \equiv P'}{(\nu m)P \equiv (\nu m)P'}$$

Now we can complete the formal description of the reaction relation. We let the *reaction relation*, $\to$, be the least relation on closed processes that satisfies $\overline{m}\langle N\rangle.P \mid m(x).Q \to P \mid Q[N/x]$ and the following rules:

$$\frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q}$$

$$\frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \frac{P \to P'}{(\nu n)P \to (\nu n)P'}$$

This definition of the reaction relation corresponds to the informal description of process behaviour given in Sections 2.2 and 3.1.

As an example, we can use the definition of the reaction relation to show the behaviour of the protocol of Section 3.2.2:

$$\begin{array}{rcl}
Inst(M) & \equiv & (\nu K_{AS})(\nu K_{SB})(A(M) \mid S \mid B) \\
& \to & (\nu K_{AS})(\nu K_{SB})(\nu K_{AB}) \\
& & \quad (\overline{c_{AB}}\langle\{M\}_{K_{AB}}\rangle \mid \overline{c_{SB}}\langle\{K_{AB}\}_{K_{SB}}\rangle \mid B) \\
& \to & (\nu K_{AS})(\nu K_{SB})(\nu K_{AB}) \\
& & \quad (\overline{c_{AB}}\langle\{M\}_{K_{AB}}\rangle \mid \\
& & \quad c_{AB}(z).case\ z\ of\ \{w\}_{K_{AB}}\ in\ F(w)) \\
& \to & (\nu K_{AS})(\nu K_{SB})(\nu K_{AB})F(M) \\
& \equiv & F(M)
\end{array}$$

The last step in this calculation is justified by our general convention that none of the bound parameters of the protocol (including, in this case, K_{AS}, K_{SB}, and K_{AB}) occurs free in F.

4.2 Testing Equivalence

In order to define equivalence, we first define a predicate that describes the channels on which a process can communicate. We let a *barb*, β, be an input or output channel, that is, either a name m (representing input) or a *co-name* $\overline{m}$ (representing output). For a closed process P, we define the predicate P *exhibits barb* β, written $P \downarrow \beta$, by the two axioms:

$$m(x).P \downarrow m \qquad\qquad \overline{m}\langle M\rangle.P \downarrow \overline{m}$$

and the three rules:

$$\frac{P \downarrow \beta}{P \mid Q \downarrow \beta} \qquad \frac{P \downarrow \beta \quad \beta \notin \{m, \overline{m}\}}{(\nu m)P \downarrow \beta}$$

$$\frac{P \equiv Q \quad Q \downarrow \beta}{P \downarrow \beta}$$

Intuitively, $P \downarrow \beta$ holds just if P is a closed process that may input or output immediately on barb β. The *convergence* predicate $P \Downarrow \beta$ holds if P is a closed process that exhibits β after some reactions:

$$\frac{P \downarrow \beta}{P \Downarrow \beta} \qquad \frac{P \to Q \qquad Q \Downarrow \beta}{P \Downarrow \beta}$$

We let a *test* consist of any closed process R and any barb β. A closed process P *passes* the test if and only if $(P \mid R) \Downarrow \beta$. The notion of testing gives rise to a testing equivalence on the set *Proc* of closed processes:

$$\begin{array}{lcl} P \simeq Q & \triangleq & \text{for any test } (R, \beta), \\ & & (P \mid R) \Downarrow \beta \text{ if and only if } (Q \mid R) \Downarrow \beta \end{array}$$

The idea of testing equivalence comes from the work of De Nicola and Hennessy [DH84]. Despite superficial differences, we can show that our relation $\simeq$ is a version of De Nicola and Hennessy's may-testing equivalence. As De Nicola and Hennessy have explained, may-testing corresponds to partial correctness (or safety), while must-testing corresponds to total correctness. Like much of the security literature, our work focuses on safety properties, hence our definitions.

A test neatly formalises the idea of a generic experiment or observation another process (such as an attacker) might perform on a process, so testing equivalence captures the concept of equivalence in an arbitrary environment. One possible drawback of testing equivalence is that it is sensitive to the choice of language [BN95]. However, our results appear fairly robust in that they carry over smoothly to some extensions of our calculus.

5 Further Cryptographic Primitives

Although so far we have discussed only shared-key cryptography, other kinds of cryptography are also easy to treat within the spi calculus. In this section we show how to handle cryptographic hashing, public-key encryption, and digital signatures. We add syntax for these operations to the spi calculus and give their semantics. We thus provide evidence that our ideas are applicable to a wide range of security protocols, beyond those that rely on shared-key encryption. We believe that we may be able to deal similarly with Diffie-Hellman techniques and with secret sharing. However, protocols for oblivious transfer and for zero-knowledge proofs, for example, are probably beyond the scope of our approach.

5.1 Hashing

A cryptographic hash function has the properties that it is very expensive to recover an input from its image or to find two inputs with the same image. Functions such as SHA and RIPE-MD are generally believed to have these properties [Sch94].

When we represent hash functions in the spi calculus, we pretend that operations that are very expensive are altogether impossible. We simply add a construct to the syntax of terms of the spi calculus:

$L, M, N ::=$	terms	
$\quad \ldots$		as in Section 3.1
$\quad H(M)$		hashing

The syntax of processes is unchanged. Intuitively, $H(M)$ represents the hash of M. The absence of a construct for recovering M from $H(M)$ corresponds to the assumption that H cannot be inverted. The lack of any equations $H(M) = H(M')$ corresponds to the assumption that H is free of collisions.

5.2 Public-Key Encryption and Digital Signatures

Traditional public-key encryption systems are based on key pairs. Normally, one of the keys in each pair is private to one principal, while the other key is public. Any principal can encrypt a message using the public key; only a principal that has the private key can then decrypt the message [DH76, RSA78].

We assume that neither key can be recovered from the other. We could just as easily deal with the case where the public key can be derived from the private one. Much as in Section 3.1, we also assume that the only way to decrypt an encrypted packet is to know the corresponding private key; that an encrypted packet does not reveal the public key that was used to encrypt it; and that there is sufficient redundancy in messages so that the decryption algorithm can detect whether a ciphertext was encrypted with the expected public key.

We arrive at the following syntax for the spi calculus with public-key encryption. (This syntax is concise, rather than memorable.)

$L, M, N ::=$	terms	
$\quad \ldots$		as in Section 3.1
$\quad M^+$		public part
$\quad M^-$		private part
$\quad \{[M]\}_N$		public-key encryption
$P, Q ::=$	processes	
$\quad \ldots$		as in Section 3.1
$\quad$ *case* L *of* $\{[x]\}_N$ *in* P		decryption

If M represents a key pair, then M^+ represents its public half and M^- represents its private half. Given a public key N, the term $\{[M]\}_N$ represents the result of the public-key encryption of M with N. In *case* L *of* $\{[x]\}_N$ *in* P, the variable x is bound in P. This construct is useful when N is a private key K^-; then it binds x to the M such that $\{[M]\}_{K^+}$ is L, if such an M exists.

It is also common to use key pairs for digital signatures. Private keys are used for signing, while public keys are used for checking signatures. We can represent digital signatures through the following extended syntax:

$L, M, N ::=$	terms	
$\ldots$		as above
$[\{M\}]_N$		private-key signature
$P, Q ::=$	processes	
$\ldots$		as above
$\mathit{case}\ N\ \mathit{of}\ [\{x\}]_M\ \mathit{in}\ P$		signature check

Given a private key N, the term $[\{M\}]_N$ represents the result of the signature of M with N. Again, the variable x is bound in P in the construct $\mathit{case}\ N\ \mathit{of}\ [\{x\}]_M\ \mathit{in}\ P$. This construct is dual to $\mathit{case}\ L\ \mathit{of}\ \{[x]\}_N\ \mathit{in}\ P$. The new construct is useful when N is a public key K^+; then it binds x to the M such that $[\{M\}]_{K^-}$ is L, if such an M exists. (Thus, we are assuming that M can be recovered from the result of signing it; but there is no difficulty in dropping this assumption.)

Formally, the semantics of the new constructs is captured with two new rules for the reduction relation:

$$
\begin{array}{lcl}
\mathit{case}\ \{[M]\}_{N^+}\ \mathit{of}\ \{[x]\}_{N^-}\ \mathit{in}\ P & > & P[M/x] \\
\mathit{case}\ [\{M\}]_{N^-}\ \mathit{of}\ [\{x\}]_{N^+}\ \mathit{in}\ P & > & P[M/x]
\end{array}
$$

As a small example, we can write the following public-key analogue for the protocol of Section 3.2.1:

$$
\begin{array}{rcl}
A(M) & \triangleq & \overline{c_{AB}}\langle\{[M, [\{H(M)\}]_{K_A^-}]\}_{K_B^+}\rangle \\
B & \triangleq & c_{AB}(x).\mathit{case}\ x\ \mathit{of}\ \{[y]\}_{K_B^-}\ \mathit{in} \\
& & \mathit{let}\ (y_1, y_2) = y\ \mathit{in} \\
& & \mathit{case}\ y_2\ \mathit{of}\ [\{z\}]_{K_A^+}\ \mathit{in} \\
& & [H(y_1)\ \mathit{is}\ z]\ F(y_1) \\
\mathit{Inst}(M) & \triangleq & (\nu K_A)(\nu K_B)(A(M) \mid B)
\end{array}
$$

In this protocol, A sends M on the channel c_{AB}, signed with A's private key and encrypted under B's public key; the signature is applied to a hash of M rather than to M itself. On receipt of a message on c_{AB}, B decrypts using its private key, checks A's signature using A's public key, checks the hash, and applies F to the body of the message (to M). The key pairs K_A and K_B are restricted; but there would be no harm in sending their public parts K_A^+ and K_B^+ on a public channel.

Undoubtedly, other formalisations of public-key cryptography are possible, perhaps even desirable. In particular, we have represented cryptographic operations at an abstract level, and do not attempt to model closely the properties of any one algorithm. We are concerned with public-key encryption and digital signatures in general rather than with their RSA implementations, say. The RSA system satisfies equations that our formalisation does not capture. For example, in the RSA system, $[\{\{[M]\}_{K^+}\}]_{K^-}$ equals M. We leave the treatment of those equations for future work.

6 Conclusions

We have applied both the standard pi calculus and the new spi calculus in the description and analysis of security protocols. We showed how to represent protocols and how to express their security properties. Our model of protocols takes into account the possibility of attacks, but does not require writing explicit specifications for an attacker. In particular, we express secrecy properties as simple equations that mean indistinguishability from the point of view of an arbitrary attacker. To our knowledge, this sharp treatment of attacks has not been previously possible.

As examples, we chose protocols of the sort commonly found in the authentication literature. Although our examples are small, we have found them instructive and encouraging. In particular, there seems to be no fundamental difficulty in writing other kinds of examples, such as protocols for electronic commerce. Unfortunately, the specifications for those protocols do not yet seem to be fully understood, even in informal terms [Mao96].

In both the pi calculus and the spi calculus, restriction and scope extrusion play a central role. The pi calculus provides an abstract treatment of channels, while the spi calculus expresses the cryptographic operations that usually underlie channels in systems for distributed security. Thus, the pi calculus and the spi calculus are appropriate at different levels. Still, it should be possible and useful to relate those levels, enabling the formal development of cryptographic protocols from non-cryptographic specifications.

Acknowledgements

Peter Sewell and Phil Wadler suggested improvements to a draft of this paper.

References

[ABLP93] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[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.

[BB90] G. Berry and G. Boudol. The chemical abstract machine. In *Conference Record of the Seventeenth ACM Symposium on Principles of Programming Languages*, pages 81–94, 1990.

[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.

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

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

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

[DY81] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. 22th IEEE Symposium on Foundations of Computer Science*, pages 350–357, 1981.

[GM95] J. Gray and J. McLean. Using temporal logic to specify and verify cryptographic protocols (progress report). In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 108–116, 1995.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[Kem89] R. A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7, 1989.

[LABW92] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.

[Lie93] A. Liebl. Authentication in distributed systems: A bibliography. *ACM Operating Systems Review*, 27(4):31–41, 1993.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996.

[Mao96] W. Mao. On two proposals for on-line bankcard payments using open networks: Problems and solutions. In *IEEE Symposium on Security and Privacy*, pages 201–210, 1996.

[MCF87] J. K. Millen, S. C. Clark, and S. B. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, February 1987.

[Mea92] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–36, 1992.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall International, 1989.

[Mil92] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2:119–141, 1992.

[Mil95] J. K. Millen. The Interrogator model. In *IEEE Symposium on Security and Privacy*, pages 251–260, 1995.

[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.

[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.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sch94] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., 1994.

[Sch96] S. Schneider. Security properties and CSP. In *IEEE Symposium on Security and Privacy*, pages 174–187, 1996.