Reasoning about Cryptographic Protocols in the Spi Calculus

Martín Abadi[1] and Andrew D. Gordon[2]

[1] Digital Equipment Corporation, Systems Research Center
[2] University of Cambridge, Computer Laboratory

Abstract. The spi calculus is an extension of the pi calculus with constructs for encryption and decryption. This paper develops the theory of the spi calculus, focusing on techniques for establishing testing equivalence, and applying these techniques to the proof of authenticity and secrecy properties of cryptographic protocols.

1 From Cryptography to Testing Equivalence

The idea of controlling communication by capabilities underlies both the pi calculus and much of the current work on security in distributed systems (see e.g. [MPW92, Lie93, Sch96b]). In the pi calculus, channel names are capabilities; a process can use a channel only if it has invented or been given the name of the channel, but cannot guess this name. In work on security, on the other hand, the capabilities for communication are often keys, which are used for encrypting and decrypting messages that travel on otherwise unprotected channels.

These observations motivate the definition of the spi calculus, an extension of the pi calculus with constructs for encryption and decryption. In a recent paper [AG97a], we introduced the spi calculus and we showed how it can be used for describing protocols, particularly authentication protocols. This paper develops the theory of the spi calculus, concentrating on results and techniques for verifying security properties of protocols.

As a first, informal example, let us consider a protocol where a user A sends a message M under a shared key K to a user B on a public channel. It is straightforward to write this protocol in the spi calculus. We may want to formalise, and then verify, two important properties of this protocol: (1) if B receives a message, then it is the one that A sent, and (2) no eavesdropper learns M. These properties hold even in the presence of an active attacker, provided the attacker does not have access to K. Taking the point of view of such an attacker, we can rephrase properties (1) and (2) in terms of an informal notion of indistinguishability: (1) the protocol is indistinguishable from one where B discards the message that it receives and acts as though the message was M, and (2) assuming that A and B do not reveal M beforehand or afterwards, the protocol is indistinguishable from one where some other message M' is sent instead of M.

In light of this example, it seems important to find a formal counterpart to the informal notion of indistinguishability. The concurrency literature suggests several candidates. Unfortunately, a straightforward notion of bisimilarity would be too fine-grained (because it would be sensitive to the contents of encrypted messages even if the corresponding keys are never disclosed). A more promising approach is based on testing equivalence. Roughly, two processes P and Q are testing equivalent if, although they may have different internal structure, a third process R cannot differentiate running in parallel with P from running in parallel with Q. We like testing equivalence for the specification of security properties because it is sufficiently coarse-grained, and because a test captures the idea of an observation that another process (such as an attacker) may perform on a process.

Our testing equivalence is a version of De Nicola's and Hennessy's may-testing equivalence [DH84], which essentially relies on tests of safety properties. Our emphasis on safety agrees with that of the security literature, where liveness is often secondary (in part because it is typically hard to guarantee).

While the use of testing equivalence may lead to neat specifications, it also implies serious difficulties in verifications. Naively, in order to prove that two processes P and Q are testing equivalent, one would need to consider an arbitrary attacker R and arbitrary sequences of interactions with R. This paper investigates more sophisticated methods for proving testing equivalence in the spi calculus. Some of these methods are based on earlier work on the pi calculus, with new ingredients motivated by our interest in security and cryptography. While our techniques may not be complete, they constitute a useful set of tools. With these techniques, we can establish security properties of particular cryptographic protocols, for example formal versions of properties (1) and (2) discussed above.

Contents

Section 2 is an informal review of the spi calculus. Section 3 gives two operational semantics of the spi calculus. Section 4 defines testing equivalence precisely and commences its study. Section 5 and 6 study some other equivalences and the new underpinning relation; these are useful in our proofs of testing equivalence. Section 7 discusses how our techniques can be applied, through an example. Finally, Section 8 mentions some related work and some conclusions. Throughout, we restrict attention to shared-key cryptography (where the same key is used for encrypting and decrypting a message); there are variants of the spi calculus for other flavours of cryptography, but we have not yet studied their theory.

This paper is a companion to the paper where we introduced the spi calculus [AG97a]. That paper covers the material of Section 2 at a more leisurely pace, with a gentle introduction to the pi calculus and the spi calculus, and many examples. It also explains the simpler of the operational semantics (the reaction relation), then gives the definition of testing equivalence. It does not contain theorems, proof techniques, or proofs for examples, which are the subject of this paper. Most of the material in the two papers is included in a longer report [AG97b].

2 The Spi Calculus (Informal Summary)

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.

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

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

In the standard pi calculus, names are the only terms. We have added constructs for pairing and numbers, (M, N), 0, and $suc(M)$, so as to permit a direct treatment of protocols that use these constructs. Moreover, we distinguish variables from names; this distinction simplifies the treatment of some equivalences. We have also added a construct for encryption (avoiding the complexity of possible encodings within the pi calculus). Intuitively, $\{M\}_N$ represents the ciphertext obtained by encrypting the term M under the key N using a shared-key cryptosystem such as DES [DES77].

The 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
$[M \textit{ is } N]\ P$	match
$\mathbf{0}$	nil
$\textit{let } (x, y) = M \textit{ in } P$	pair splitting
$\textit{case } M \textit{ of } 0 : P\ suc(x) : Q$	integer case
$\textit{case } L \textit{ of } \{x\}_N \textit{ in } P$	shared-key decryption

In $M(x).P$, the variable x is bound in P. In $(\nu n)P$, the name n is bound in P. In $\textit{let } (x, y) = M \textit{ in } P$, the variables x and y are bound in P. In $\textit{case } M \textit{ of } 0 : P\ suc(x) : Q$, the variable x is bound in Q. In $\textit{case } L \textit{ of } \{x\}_N \textit{ in } P$, the variable x is bound in P. 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 abbreviations $\prod_{i \in 1..k} P_i$ for the k-way parallel composition $P_1 \mid \cdots \mid P_k$, and $\overline{M}\langle N\rangle$ for $\overline{M}\langle N\rangle.\mathbf{0}$ and $M(x)$ for $M(x).\mathbf{0}$. We also express the k-fold restriction $(\nu p_1) \cdots (\nu p_k)P$ by $(\nu p_i{}^{i \in 1..k})P$ or, when $\vec{p} = p_1, \ldots, p_k$, by $(\nu\vec{p})P$. Intuitively, processes have the following meanings:

- An *output process* $\overline{m}\langle N\rangle.P$ is ready to output N on m, and then to behave as P. The output happens only when there is a process ready to input from

m. An *input process* $m(x).Q$ is ready to input from m, and then to behave as $Q[N/x]$, where N is the input received.
The general forms $\overline{M}\langle N\rangle.P$ and $M(x).Q$ 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 P and Q running in parallel.
- 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 infinitely many copies of P running in parallel.
- A *match* $[M \textit{ is } N]\, P$ behaves as P provided that M and N are the same term; otherwise it is stuck, that is, it does nothing.
- The *nil* process $\mathbf{0}$ does nothing.
- A *pair splitting* process $\textit{let } (x, y) = M \textit{ in } P$ behaves as $P[N/x][L/y]$ if M is a pair (N, L), and it is stuck if M is not a pair.
- An *integer case* process $\textit{case } M \textit{ of } 0 : P\ \textit{suc}(x) : Q$ behaves as P if M is 0, as $Q[N/x]$ if M is $\textit{suc}(N)$ for some N, and otherwise is stuck.
- A *decryption* process $\textit{case } L \textit{ of } \{x\}_N \textit{ in } P$ attempts to decrypt L with the key N. If L has the form $\{M\}_N$, then the process behaves as $P[M/x]$. Otherwise the process is stuck.

For example, $(\nu K)\overline{m}\langle\{0\}_K\rangle$ is a process that creates a name K and then sends the result of encrypting the numeral 0 with K on a channel m. Here we use the letter K for a name because the name plays the role of a key. Since m is not bound, anyone may receive $\{0\}_K$; however, since K is bound, this term cannot be successfully decrypted. In order to illustrate the use of decryption, we add a process that has the necessary key:

$$(\nu K)(\overline{m}\langle\{0\}_K\rangle \mid m(y).\textit{case } y \textit{ of } \{x\}_K \textit{ in } \overline{m}\langle x\rangle)$$

The new process (to the right of $\mid$) tries to decrypt the message y that it receives on m using K, and sends the result x on m.

We write $\textit{fn}(M)$ and $\textit{fn}(P)$ for the sets of names free in term M and process P respectively, and write $\textit{fv}(M)$ and $\textit{fv}(P)$ for the sets of variables free in M and P respectively. A term or process is *closed* if it has no free variables; *Proc* is the set of closed processes.

A context $\mathcal{C}$ is a process with a single hole; $\mathcal{C}[P]$ is the outcome of filling the hole with P. A congruence is an equivalence relation $\mathcal{R}$ such that if $P \,\mathcal{R}\, Q$ then $\mathcal{C}[P] \,\mathcal{R}\, \mathcal{C}[Q]$ for any context $\mathcal{C}$. One can restrict attention to closed processes and contexts, but the claims of this paper hold independently of this restriction.

3 Two Operational Semantics

The reaction relation is a concise account of computation in the pi calculus introduced by Milner [Mil92], with ideas from Berry and Boudol [BB92]. In this section we adapt the reaction relation to the spi calculus.

The definition of reaction is rather elegant, but not convenient for proofs (because it relies on an auxiliary notion of structural equivalence). Therefore, we provide an alternative, direct characterisation of reaction. We achieve this by defining a commitment relation, in the style of Milner [Mil95b].

The Reaction Relation

The definition of reaction has several phases. First we define a *reduction relation* $>$ on *Proc*. We require that $!P > (P \mid !P)$, that $[M \textit{ is } M]\, P > P$, and that $\textit{case } \{M\}_N \textit{ of } \{x\}_N \textit{ in } P > P[M/x]$; we omit similar conditions for *let* and for the *case* construct for numerals. Then we define a *structural equivalence relation* $\equiv$ on *Proc*. It is the least equivalence relation including $>$ such that $P \equiv P'$ implies $P \mid Q \equiv P' \mid Q$ and $(\nu m)P \equiv (\nu m)P'$, and such that

$$\begin{array}{ccc} P \mid \mathbf{0} \equiv P & P \mid Q \equiv Q \mid P & P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\ (\nu m)(\nu n)P \equiv (\nu n)(\nu m)P & (\nu n)\mathbf{0} \equiv \mathbf{0} & ((\nu n)P) \mid Q \equiv (\nu n)(P \mid Q) \end{array}$$

where, in the last equation, $n \notin \textit{fn}(Q)$. Intuitively, this equivalence relation allows processes to be rearranged so that reaction is possible. Finally, the *reaction relation* $\rightarrow$ is the least relation on *Proc* such that $P \rightarrow P'$ implies $P \mid Q \rightarrow P' \mid Q$ and $(\nu m)P \rightarrow (\nu m)P'$, such that $P \equiv P' \rightarrow Q' \equiv Q$ implies $P \rightarrow Q$, and such that $\overline{m}\langle N \rangle.P \mid m(x).Q \rightarrow P \mid Q[N/x]$.

(The reduction relation, $>$, is not found in previous accounts of the pi calculus. We introduced it in the definition of structural equivalence because it is useful also in the definition of commitment, given next.)

The Commitment Relation

In order to define commitment, we need some new syntactic forms. An *abstraction* is an expression of the form $(x)P$, where x is a bound variable and P is a process. Intuitively, $(x)P$ is like the process $p(x).P$ minus the name p. When F is the abstraction $(x)P$ and M is a term, we write $F(M)$ for $P[M/x]$. A *concretion* is an expression of the form $(\nu m_1, \ldots, m_k)\langle M \rangle P$, where M is a term, P is a process, $k \geq 0$, and the names $m_1, \ldots, m_k$ are bound in M and P. Intuitively, $(\nu m_1, \ldots, m_k)\langle M \rangle P$ is like the process $(\nu m_1) \ldots (\nu m_k)\overline{p}\langle M \rangle P$ minus the name p, provided p is not one of $m_1, \ldots, m_k$. We often write concretions as $(\nu \vec{m})\langle M \rangle P$, where $\vec{m} = m_1, \ldots, m_k$, or simply $(\nu)\langle M \rangle P$ if $k = 0$. Finally, an *agent* is an abstraction, a process, or a concretion. We use the metavariables A and B to stand for arbitrary agents, and let $\textit{fv}(A)$ and $\textit{fn}(A)$ be the sets of free variables and free names of an agent A, respectively.

We extend the restriction and composition operators to arbitrary agents, as follows. For an abstraction $(x)P$, we let $(\nu m)(x)P = (x)(\nu m)P$ and let $R \mid (x)P = (x)(R \mid P)$, assuming that $x \notin \textit{fv}(R)$. For a concretion $(\nu \vec{n})\langle M \rangle Q$, we let $(\nu m)(\nu \vec{n})\langle M \rangle Q = (\nu m, \vec{n})\langle M \rangle Q$ if $m \in \textit{fn}(M)$ and $= (\nu \vec{n})\langle M \rangle (\nu m)Q$ otherwise, and let $R \mid (\nu \vec{n})\langle M \rangle Q = (\nu \vec{n})\langle M \rangle (R \mid Q)$, in all cases assuming that $m \notin \{\vec{n}\}$ and that $\{\vec{n}\} \cap \textit{fn}(R) = \emptyset$. We define the dual composition $A \mid R$ symmetrically. If F

is the abstraction $(x)P$ and C is the concretion $(\nu\vec{n})\langle M\rangle Q$, and $\{\vec{n}\}\cap fn(P) = \emptyset$, we define the *interactions* $F@C$ and $C@F$ to be the processes given by:

$$F@C \triangleq (\nu\vec{n})(P[M/x] \mid Q) \qquad C@F \triangleq (\nu\vec{n})(Q \mid P[M/x])$$

Intuitively, these processes are the possible immediate results of the encounter of F and C. Given a common name p, we have that F is like $p(x).P$ and C is like $(\nu\vec{n})\overline{p}\langle M\rangle P$, so an interaction of F and C is a process obtained when $p(x).P$ and $(\nu\vec{n})\overline{p}\langle M\rangle P$, put in parallel, communicate on p.

An *action* is a name m (representing input), a co-name $\overline{m}$ (representing output), 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 inductively by the axioms $m(x).P \xrightarrow{m} (x)P$ and $\overline{m}\langle M\rangle.P \xrightarrow{\overline{m}} (\nu)\langle M\rangle P$, and the rules:

$$\frac{P \xrightarrow{m} F \quad Q \xrightarrow{\overline{m}} C}{P \mid Q \xrightarrow{\tau} F@C} \qquad \frac{P \xrightarrow{\overline{m}} C \quad Q \xrightarrow{m} F}{P \mid Q \xrightarrow{\tau} C@F} \qquad \frac{P > Q \quad Q \xrightarrow{\alpha} A}{P \xrightarrow{\alpha} A}$$

$$\frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q} \qquad \frac{Q \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} P \mid A} \qquad \frac{P \xrightarrow{\alpha} A \quad \alpha \notin \{m, \overline{m}\}}{(\nu m)P \xrightarrow{\alpha} (\nu m)A}$$

The commitment relation indexed by τ, $\xrightarrow{\tau}$, is a binary relation on *Proc*. We write $\xrightarrow{\tau}^*$ for its reflexive and transitive closure; and we write $P \xrightarrow{\tau}\equiv Q$ when there exists a process R such that $P \xrightarrow{\tau} R$ and $R \equiv Q$.

Proposition 1. *$P \to Q$ if and only if $P \xrightarrow{\tau}\equiv Q$.*

4 Testing Equivalence

As explained in the introduction, we rely on testing equivalence for the specification of security properties. This section gives the definition of tests and testing equivalence, and some basic results about them.

A *barb* is either a name m or a co-name $\overline{m}$. If P is a closed process, we say that P *exhibits barb* β, and write $P \downarrow \beta$, if P may communicate immediately on β. Similarly, the *convergence* predicate $P \Downarrow \beta$ means that P may communicate on β after some reactions. Formally, these two predicates are defined by the axioms $m(x).P \downarrow m$ and $\overline{m}\langle M\rangle.P \downarrow \overline{m}$, and by the rules:

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

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

A *test*, (R, β), consists of a closed process R and a barb β. A closed process P *passes* the test if and only if $(P \mid R) \Downarrow \beta$. The following proposition enables us to use the commitment relation in proofs about tests:

Proposition 2. *$P \downarrow \beta$ if and only if $P \xrightarrow{\beta} A$ for some agent A. Moreover, P passes a test (R, β) if and only if*

$$P \mid R \xrightarrow{\tau}{}^{*} Q \textit{ and } Q \xrightarrow{\beta} A$$

for some agent A and process Q.

The definition of testing induces a testing preorder $\sqsubseteq$ and a testing equivalence $\simeq$ on *Proc*:

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

The following proposition is crucial for proofs of testing equivalence:

Proposition 3. *Testing equivalence is a congruence. Furthermore, structural equivalence implies testing equivalence.*

We have developed a number of auxiliary lemmas about testing equivalence, for use in our verifications. As a sample, we show a proposition that enables us to reduce infinite problems (with replication) to finite but unbounded problems:

Proposition 4. *If $(\nu\vec{p})(P_1 \mid \prod_{i \in 1..n} P_2) \simeq (\nu\vec{p})(Q_1 \mid \prod_{i \in 1..n} Q_2)$ for all $n \geq 0$, then $(\nu\vec{p})(P_1 \mid !P_2) \simeq (\nu\vec{p})(Q_1 \mid !Q_2)$.*

Testing equivalence becomes finer-grained (and perhaps even more compelling) in the presence of a construct $[M \textit{ is not } N]\ P$ for mismatch [BN95]. Our calculus does not include that construct because we have not found a need for it in writing protocols. The same is true for other sensible negative constructs, for example one that checks whether a term is not a pair. We believe that our results remain valid in the presence of such constructs.

5 Other Equivalences Useful in Proofs

In addition to testing equivalence, several other equivalences are useful for reasoning about cryptographic protocols. In particular, barbed congruence is an equivalence with a tractable co-inductive presentation and is stronger than testing equivalence. The definitions of these equivalences are adaptations of corresponding definitions in the pi calculus. However, the presence of encryption affects the use of these equivalences in examples.

Strong Bisimilarity

If $\mathcal{R}$ is a relation on closed processes, we define the relation $\overline{\mathcal{R}}$ on closed agents:

$$\begin{array}{ll} P \,\overline{\mathcal{R}}\, Q & \text{iff } P \,\mathcal{R}\, Q \\ (x)P \,\overline{\mathcal{R}}\, (x)Q & \text{iff } P[M/x] \,\mathcal{R}\, Q[M/x] \text{ for all closed } M \\ (\nu\vec{n})\langle M\rangle P \,\overline{\mathcal{R}}\, (\nu\vec{m})\langle M\rangle Q & \text{iff } \vec{m} \text{ is a permutation of } \vec{n} \text{ and } P \,\mathcal{R}\, Q \end{array}$$

A *strong simulation* is a binary relation $\mathcal{S}$ on *Proc* such that if $P\ \mathcal{S}\ Q$ and $P \xrightarrow{\alpha} A$ then there exists B with $Q \xrightarrow{\alpha} B$ and $A\ \overline{\mathcal{S}}\ B$. A relation $\mathcal{S}$ is a *strong bisimulation* if and only if both $\mathcal{S}$ and its converse $\mathcal{S}^{-1}$ are strong simulations [Mil95b]. *Strong bisimilarity*, written $\sim_s$, is the greatest strong bisimulation, namely the union of all strong bisimulations.

Strong bisimilarity is a rather fine-grained equivalence for the spi calculus. For instance, it discriminates between the processes $(\nu K)\overline{c}\langle\{M\}_K\rangle$ and $(\nu K)\overline{c}\langle\{M'\}_K\rangle$, which we would wish to equate since these processes send encrypted messages under keys that are never disclosed, so their difference should not matter. Still, strong bisimilarity is often useful in justifying particular steps of our proofs.

Barbed Equivalence

Intuitively, one way of weakening strong bisimilarity is to ignore what messages are sent on what channels, and to record only what channels are used. This informal idea leads to the following definition. A *barbed simulation* is a binary relation $\mathcal{S}$ on *Proc* such that $P\ \mathcal{S}\ Q$ implies:

(1) for each barb β, if $P \downarrow \beta$ then $Q \downarrow \beta$, and
(2) if $P \rightarrow P'$ then there exists Q' such that $Q \rightarrow Q'$ and $P' \equiv\mathcal{S}\equiv Q'$

where $P' \equiv\mathcal{S}\equiv Q'$ means that there exist P'' and Q'' such that $P' \equiv P''$, $P''\ \mathcal{S}\ Q''$, and $Q'' \equiv Q'$. A *barbed bisimulation* is a relation $\mathcal{S}$ such that both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are barbed simulations. *Barbed equivalence*, written $\dot{\sim}$, is the greatest barbed bisimulation.

Proposition 5. *Barbed equivalence is an equivalence relation, and is preserved by restriction. Furthermore, both structural equivalence and strong bisimilarity imply barbed equivalence.*

In order to establish a barbed equivalence, it is often convenient to use Milner's technique of "bisimulation up to" [Mil89, MPW92]. A *barbed simulation up to $\dot{\sim}$ and restriction* is a binary relation $\mathcal{S}$ on *Proc* such that $P\ \mathcal{S}\ Q$ implies:

(1) for each barb β, if $P \downarrow \beta$ then $Q \downarrow \beta$, and
(2) if $P \rightarrow P'$ then there exists Q' such that $Q \rightarrow Q'$, and there exist P'', Q'', and names $\vec{n}$ such that $P' \mathrel{\dot{\sim}} (\nu\vec{n})P''$, $Q' \mathrel{\dot{\sim}} (\nu\vec{n})Q''$, and $P''\ \mathcal{S}\ Q''$.

A *barbed bisimulation up to $\dot{\sim}$ and restriction* is a relation $\mathcal{S}$ such that both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are barbed simulations up to $\dot{\sim}$ and restriction.

Proposition 6. *If $\mathcal{S}$ is a barbed bisimulation up to $\dot{\sim}$ and restriction, then $\mathcal{S} \subseteq \dot{\sim}$.*

Barbed equivalence is still only a stepping stone. One reason for this is that there are processes that are barbed equivalent but not strongly bisimilar or testing equivalent, such as $\overline{m}\langle n\rangle.\overline{m}\langle n\rangle$ and $\overline{m}\langle n\rangle$. Moreover, barbed equivalence is far from being a congruence: it is not even closed under composition, as can be seen by comparing $\overline{m}\langle n\rangle.\overline{m}\langle n\rangle \mid m(x)$ and $\overline{m}\langle n\rangle \mid m(x)$.

Barbed Congruence

Barbed congruence, written $\sim$, is the relation on *Proc* obtained by strengthening barbed equivalence as follows:

$$P \sim Q \;\triangleq\; \forall R \in \mathit{Proc}(P \mid R \mathrel{\dot\sim} P \mid R)$$

For example, it is easy to prove that a process that attempts to decrypt "with the wrong key" is barbed congruent to $\mathbf{0}$:

$$\mathit{case}\ M\ \mathit{of}\ \{x\}_K\ \mathit{in}\ P \;\sim\; \begin{cases} P[N/x] & \text{if } M = \{N\}_K \text{ for some } N \\ \mathbf{0} & \text{otherwise} \end{cases}$$

This is an example of the fairly obvious fact that barbed congruence does not imply structural equivalence.

As another example, we prove in Section 6 that $(\nu K)\overline{c}\langle\{M\}_K\rangle$ is barbed congruent to $(\nu K)\overline{c}\langle\{M'\}_K\rangle$. This example shows that barbed congruence does not imply strong bisimilarity. It also shows that barbed congruence is not sensitive to the contents of encrypted messages whose keys are not disclosed, suggesting that barbed congruence is coarse-grained enough for interesting cryptographic applications.

Proposition 7. *Barbed congruence is a congruence. Furthermore, both structural equivalence and strong bisimilarity imply barbed congruence, while barbed congruence implies testing equivalence.*

Thus, whenever we wish to prove a testing equivalence, we may try to prove a barbed congruence, as a sufficient condition. However, testing equivalence does not imply barbed congruence, because barbed congruence is more sensitive to τ steps and to branching structure.

6 The Underpinning Relation

This section introduces the *underpinning relation*, which is also useful in proofs. We say that $x_1{:}\{-\}_{p_1}, \ldots, x_n{:}\{-\}_{p_n}$ *underpins* the agent A roughly if A is an agent that may contain occurrences of any of the variables $x_1, \ldots, x_n$, but no occurrences of any of the names $p_1, \ldots, p_n$. We write this:

$$x_1{:}\{-\}_{p_1}, \ldots, x_n{:}\{-\}_{p_n} \vdash A$$

Our intention is that A represents an attacker and that the variables $x_1, \ldots, x_n$ represent ciphertexts that the attacker may have intercepted encrypted under the keys $p_1, \ldots, p_n$, which the attacker does not have. (Here we take all keys to be names as this suffices for our present purposes; but the general case, where a key is an arbitrary term, could also be interesting.)

Next we give a precise definition of the underpinning relation. An *environment* is a finite list of entries of the form $x{:}\{-\}_n$, where x is a variable and n is a name; all the variables must be distinct (but the names need not be). We let $dom(E)$ be the set of variables mentioned in the entries in E, and $keys(E)$ the set of names mentioned in the entries in E. When E is an environment, M a term, and A an agent, we define:

$$\begin{array}{rcl} E \vdash M & \triangleq & fv(M) \subseteq dom(E) \text{ and } fn(M) \cap keys(E) = \emptyset \\ E \vdash A & \triangleq & fv(A) \subseteq dom(E) \text{ and } fn(A) \cap keys(E) = \emptyset \end{array}$$

The relation $\vdash$ is the underpinning relation.

When $x{:}\{-\}_n$ occurs in an environment, we intend that x stands for a ciphertext of the form $\{M\}_n$. An *E-closure* is a substitution that fixes all the variables in E to appropriate ciphertexts; more precisely, an E-closure is a substitution σ of closed ciphertexts for variables such that $E \vdash \sigma$ is derivable from the following rules:

$$\frac{}{\emptyset \vdash \emptyset} \qquad \frac{E \vdash \sigma \quad x \notin dom(E) \quad fv(M) = \emptyset}{E, x{:}\{-\}_n \vdash \sigma, \{M\}_n/x}$$

where $\emptyset$ represents the empty environment, the empty substitution, and the empty set, and $\sigma, \{M\}_n/x$ is the extension of σ that maps x to $\{M\}_n$.

To prove secrecy properties, we would like to show that a process underpinned by an environment acts uniformly no matter which ciphertexts are substituted for the variables in the environment. At first sight one might think that if $E \vdash P$, $E \vdash \sigma$, and $E \vdash \sigma'$, then $P\sigma \sim P\sigma'$ on the reasoning that, since P cannot unwrap the ciphertexts in σ or σ', it will behave the same whether closed by one or the other E-closure. This would hold were it not for the presence of matching in the language. For example, $E = x{:}\{-\}_m, y{:}\{-\}_m$, $P = [x \text{ } is \text{ } y]\,\overline{p}\langle 0\rangle$, $\sigma = [\{0\}_m/x,\ \{0\}_m/y]$, and $\sigma' = [\{0\}_m/x,\ \{suc(0)\}_m/y]$ meet the conditions above, but $P\sigma$ may output 0 whereas $P\sigma'$ is stuck. Thus, P can act contingently on the ciphertexts even though it cannot decrypt them. However, if we insist that σ and σ' be injective (that is, $x = y$ whenever $x\sigma = y\sigma$, and similarly for σ') then we obtain $P\sigma \sim P\sigma'$.

These informal arguments lead to the following results.

Lemma 8. *Suppose that $E \vdash P$ and $E \vdash \sigma$, and that σ is injective. If $P\sigma > Q'$ then there is a process Q with $E \vdash Q$, $fv(Q) \subseteq fv(P)$, $fn(Q) \subseteq fn(P)$, and $Q' = Q\sigma$ such that, whenever $E \vdash \sigma'$ and σ' is injective, $P\sigma' > Q\sigma'$. Similarly, if $P\sigma \xrightarrow{\alpha} A'$ then there is an agent A with $E \vdash A$, $fv(A) \subseteq fv(P)$, $fn(A) \subseteq fn(P)$, and $A' = A\sigma$ such that, whenever $E \vdash \sigma'$ and σ' is injective, $P\sigma' \xrightarrow{\alpha} A\sigma'$.*

Proposition 9. *Suppose that $E \vdash \sigma$ and $E \vdash \sigma'$, and that both σ and σ' are injective. Then $\mathcal{S} = \{(P\sigma, P\sigma') \mid E \vdash P\}$ is a barbed bisimulation.*

This last proposition provides an easy way to prove some equivalences, as we now demonstrate with a small proof of a secrecy property. We prove that, for any closed terms M and M':

$$(\nu K)\overline{c}\langle\{M\}_K\rangle \ \simeq\ (\nu K)\overline{c}\langle\{M'\}_K\rangle$$

By Proposition 7, we can reduce this testing equivalence to the barbed congruence $(\nu K)\overline{c}\langle\{M\}_K\rangle \sim (\nu K)\overline{c}\langle\{M'\}_K\rangle$. According to the definition of barbed congruence, it suffices to establish the barbed equivalence $((\nu K)\overline{c}\langle\{M\}_K\rangle) \mid R \dot{\sim} ((\nu K)\overline{c}\langle\{M'\}_K\rangle) \mid R$ for every closed process R. Since K is bound, we may assume that $K \notin fn(R)$. So we have the structural equivalences $((\nu K)\overline{c}\langle\{M\}_K\rangle) \mid R \equiv (\nu K)(\overline{c}\langle\{M\}_K\rangle \mid R)$ and $((\nu K)\overline{c}\langle\{M'\}_K\rangle) \mid R \equiv (\nu K)(\overline{c}\langle\{M'\}_K\rangle \mid R)$. Since barbed equivalence is an equivalence relation and since it includes structural equivalence (by Proposition 5), it suffices to show that $(\nu K)(\overline{c}\langle\{M\}_K\rangle \mid R) \dot{\sim} (\nu K)(\overline{c}\langle\{M'\}_K\rangle \mid R)$. Since restriction preserves barbed equivalence (also by Proposition 5), it suffices to prove that:

$$\overline{c}\langle\{M\}_K\rangle \mid R \;\dot{\sim}\; \overline{c}\langle\{M'\}_K\rangle \mid R$$

But this follows from Proposition 9 with $E = x{:}\{-\}_K$, $P = \overline{c}\langle x\rangle \mid R$, $\sigma = [\{M\}_K/x]$, and $\sigma' = [\{M'\}_K/x]$.

7 A Simple Example

In this section we describe and verify a two-message protocol. Protocols used in practice generally have more messages, but not many more. We focus on this short protocol in order to allow a concise exposition of our method. We have applied the same method to more complex protocols (in particular, to a seven-message protocol defined in [AG97a]).

The purpose of the protocol is to enable communication between two users or machines A and B. For simplicity, both A and B are fixed, and A sends a finite but unbounded set of messages to B. All the messages are transmitted under the same shared key K_{AB}. The messages are accompanied by nonces, which must be fresh and unpredictable quantities [AN96]; for each message M, A creates a nonce N_A and B creates a nonce N_B. Informally, the protocol is:

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

In Message 1, B challenges A with the nonce N_B. Message 1 may be spontaneous, or it may be triggered by an arbitrary message from A; the details of this do not matter. Message 2 is the transmission of M under K_{AB}. The presence of N_B proves the freshness of Message 2. The presence of N_A guarantees the distinctness of all ciphertexts.

For this example, we encode a triple (L, M, N) by a pair $(L, (M, N))$, and extend pair splitting to triples in the usual way. We also omit the inner brackets from an encrypted triple of the form $\{(N, N', N'')\}_{N'''}$, and simply write $\{N, N', N''\}_{N'''}$.

We express the protocol with the following parameterised definitions:

$$\begin{array}{rcl} A(N_A, M) & \triangleq & c_A(x).\overline{c_B}\langle\{N_A, x, M\}_{K_{AB}}\rangle \\ B(N_B, F) & \triangleq & \overline{c_A}\langle N_B\rangle \mid c_B(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AB}}\ \mathit{in} \\ & & \qquad \mathit{let}\ (y_1, y_2, y_3) = y\ \mathit{in} \\ & & \qquad [y_2\ \mathit{is}\ N_B]\, F(y_3) \end{array}$$

$$Sys(M_1,\ldots,M_m,F) \ \triangleq\ (\nu K_{AB})((\prod_{k\in 1..m}(\nu N_A)A(N_A,M_k)) \mid \\ !(\nu N_B)B(N_B,F))$$

The channels c_A and c_B represent input channels for A and B, respectively. Parameter F is an abstraction such that $F(M)$ is the behaviour of B when it successfully receives a message M; we assume that the bound parameters of the protocol (including in particular K_{AB}) do not occur free in F. The process $Sys(M_1,\ldots,M_m,F)$ is a description of a complete system where A sends the messages M_1, …, M_m to B, and B applies F to the messages that it receives. The restrictions model the intended scope for the shared key and for the nonces. The replication indicates that B is willing to engage in an arbitrary number of transmissions.

Intuitively, the protocol has an authenticity property: if B accepts a message, then it is one sent by A; moreover, B does not accept message replays. In other words, the property is that if $Sys(M_1,\ldots,M_m,F)$ causes the execution of $F(L_1)$, …, $F(L_l)$ then $\{L_1,\ldots,L_l\}$ is a sub-multiset of $\{M_1,\ldots,M_m\}$.

This authenticity property would be fairly obvious if $Sys(M_1,\ldots,M_m,F)$ had the form $(\nu p_k{}^{k\in 1..m})(Q \mid \prod_{k\in 1..m} p_k(x).F(M_k))$, where Q is a process independent of F. The following proposition establishes that $Sys(M_1,\ldots,M_m,F)$ is in fact equivalent to a process of this form:

Proposition 10 (Authenticity). *For any number m and distinct names p_1, …, p_m there is a closed process Q with $fn(Q) \subseteq \{p_1,\ldots,p_m,c_A,c_B\}$ such that, for all closed terms M_1, …, M_m and all closed abstractions F, if $p_1,\ldots,p_m \notin fn(Sys(M_1,\ldots,M_m,F))$ then:*

$$Sys(M_1,\ldots,M_m,F) \ \simeq\ (\nu p_k{}^{k\in 1..m})(Q \mid \prod_{k\in 1..m} p_k(x).F(M_k))$$

Proof. We only sketch the proof, for brevity. It depends on an analysis of the state space of the protocol, and on most of the other propositions presented in this paper. For an arbitrary number m and distinct names p_1, …, p_m, we set:

$$Q \ \triangleq\ (\nu K_{AB})(\prod_{k\in 1..m}(\nu N_A)A(N_A,p_k) \mid !(\nu N_B)B(N_B,G))$$

where the names K_{AB}, N_A, and N_B are distinct from one another and from p_1, …, p_m, c_A, c_B, and where G is the abstraction $(x)\overline{x}\langle 0\rangle$. (The choice of 0 is somewhat arbitrary; any term with no free names or variables would do instead.) It follows that $fn(Q) = \{p_1,\ldots,p_m,c_A,c_B\}$. Consider any closed terms M_1, …, M_m, and any closed abstraction F such that none of the names p_1, …, p_m occurs free in $Sys(M_1,\ldots,M_m,F)$. We may assume that the bound name K_{AB} does not occur free in M_1, …, M_m, or F. The right-hand side of the proposition is structurally equivalent to:

$$(\nu K_{AB})(\prod_{k\in 1..m}(\nu p_k)((\nu N_A)A(N_A,p_k) \mid p_k(x).F(M_k)) \mid \\ !(\nu N_B)B(N_B,G)) \tag{1}$$

Therefore, by Proposition 4, it suffices to show that

$$(\nu K_{AB})(\prod_{k\in 1..m}(\nu N_A)A(N_A,M_k) \mid \prod_{s\in 1..r}(\nu N_B)B(N_B,F)) \tag{2}$$

and

$$(\nu K_{AB})(\prod_{k\in 1..m}(\nu p_k)((\nu N_A)A(N_A, p_k) \mid p_k(x).F(M_k)) \mid \\ \prod_{s\in 1..r}(\nu N_B)B(N_B, G)) \tag{3}$$

are testing equivalent for all $r \geq 0$. By Propositions 5 and 7 (and via several small formal manipulations), it suffices to prove that the following two processes are barbed equivalent:

$$\prod_{k\in 1..m} A(N_{Ak}, M_k) \mid \prod_{s\in 1..r} B(N_{Bs}, \tau.F) \mid R \tag{4}$$

$$(\nu p_k{}^{k\in 1..m})(\prod_{k\in 1..m}(A(N_{Ak}, p_k) \mid p_k(x).F(M_k)) \mid \prod_{s\in 1..r} B(N_{Bs}, G) \mid R) \tag{5}$$

where $N_{A1}, \ldots, N_{Am}, N_{B1}, \ldots, N_{Br}$ are distinct, fresh names, and R is an arbitrary closed process such that none of the names $K_{AB}, N_{A1}, \ldots, N_{Am}, N_{B1}, \ldots, N_{Br}, p_1, \ldots, p_m$ occurs free in R; and where $\tau.F$ is defined by $(\tau.F)(M) = (\nu m)(\overline{m}\langle 0\rangle \mid m(x).F(M))$ with m a fresh name and x a fresh variable.

The remainder of the proof consists in constructing a relation $\mathcal{S}$ such that $\equiv\mathcal{S}\equiv$ relates processes (4) and (5), and establishing that $\mathcal{S}$ is a barbed bisimulation up to $\dot{\sim}$ and restriction. It then follows that processes (4) and (5) are barbed equivalent by Proposition 6. We omit the definition of $\mathcal{S}$ and its study, which relies on Proposition 1. □

Intuitively, the protocol has a secrecy property too: if no information about the messages is given away as they are accepted, then no external observer can identify the messages. However, the observer may be able to tell how many messages are transmitted. This secrecy property is captured in the following proposition:

Proposition 11 (Secrecy). *For any pairs of closed terms (M_1, M_1'), ..., (M_m, M_m') and any closed abstraction F, if $F(M_i) \simeq F(M_i')$ for each $i \in 1..m$ then:*

$$Sys(M_1, \ldots, M_m, F) \;\simeq\; Sys(M_1', \ldots, M_m', F)$$

Proof. Proposition 10 implies that

$$Sys(M_1, \ldots, M_m, F) \;\simeq\; (\nu p_k{}^{k\in 1..m})(Q \mid \prod_{k\in 1..m} p_k(x).F(M_k))$$

and

$$Sys(M_1', \ldots, M_m', F) \;\simeq\; (\nu p_k{}^{k\in 1..m})(Q \mid \prod_{k\in 1..m} p_k(x).F(M_k'))$$

for some process Q and some names $p_1, \ldots, p_m$. Since testing equivalence is a congruence (by Proposition 3), if $F(M_i) \simeq F(M_i')$ for each $i \in 1..m$ then:

$$(\nu p_k{}^{k\in 1..m})(Q \mid \prod_{k\in 1..m} p_k(x).F(M_k)) \simeq (\nu p_k{}^{k\in 1..m})(Q \mid \prod_{k\in 1..m} p_k(x).F(M_k'))$$

The claim follows by transitivity. □

8 Conclusions

In the last few years, the importance of reasoning about cryptographic protocols has been widely recognised, and several methods have been used for this task. Those methods are based on a large variety of formal frameworks: temporal logics, modal logics, state-transition models, CSP (see e.g. [MCF87, BAN89, Kem89, Mil95a, Mea92, GM95, Low96, Sch96a]). The main emphasis of that work has been on authenticity properties.

Proofs in the spi calculus are sometimes more difficult than proofs in those earlier frameworks. The sources of this difficulty are in part the novelties and advantages of the spi-calculus approach: the expressive scoping mechanisms of the pi calculus; the treatment of secrecy properties; the view of attacks as tests, and the focus on testing equivalence. We hope that, through techniques such as ours, proofs in the spi calculus will become easier, even routine.

Acknowledgements

Paola Quaglia, Steve Schneider, and Peter Sewell suggested improvements to a draft of this paper.

References

[AG97a] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. To appear in the Proceedings of the Fourth ACM Conference on Computer and Communications Security, 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.

[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.

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, April 1992.

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

[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.

[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.

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

[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.

[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.

[Mil95a] J. K. Millen. The Interrogator model. In *IEEE Symposium on Security and Privacy*, pages 251–260, 1995.

[Mil95b] R. Milner. The π-calculus. Undergraduate lecture notes, Cambridge University, 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.

[Sch96a] S. Schneider. Security properties and CSP. In *IEEE Symposium on Security and Privacy*, pages 174–187, 1996.

[Sch96b] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., second edition, 1996.

This article was processed using the LaTeX macro package with LLNCS style