A Bisimulation Method for Cryptographic Protocols

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

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

Abstract. We introduce a definition of bisimulation for cryptographic protocols. The definition includes a simple and precise model of the knowledge of the environment with which a protocol interacts. Bisimulation is the basis of an effective proof technique, which yields proofs of classical security properties of protocols and also justifies certain protocol optimisations. The setting for our work is the spi calculus, an extension of the pi calculus with cryptographic primitives. We prove the soundness of the bisimulation proof technique within the spi calculus.

1 Introduction

In reasoning about a reactive system, it is necessary to consider not only the steps taken by the system but also the steps taken by its environment. In the case where the reactive system is a cryptographic protocol, the environment may well be hostile, so little can be assumed about its behaviour. Therefore, the environment may be modelled as a nondeterministic process capable of intercepting messages and of sending any message that it can construct at any point. This approach to describing the environment is fraught with difficulties; the resulting model can be somewhat arbitrary, hard to understand, and hard to reason about.

Bisimulation techniques [Par81,Mil89] provide an alternative approach. Basically, using bisimulation techniques, we can equate two systems whenever we can establish a correspondence between their steps. We do not need to describe the environment explicitly, or to analyse its possible internal computations.

Bisimulation techniques have been applied in a variety of areas and under many guises. Their application to cryptographic protocols, however, presents new challenges.

- Consider, for example, a secure communication protocol $P(M)$, where some cleartext M is transmitted encrypted under a session key. We may like to argue that $P(M)$ preserves the secrecy of M, and may want to express this secrecy property by saying that $P(M)$ and $P(N)$ are equivalent, for every M and N. This equivalence may be sensible because, although $P(M)$ and $P(N)$ send different messages, an attacker that does not have the session key cannot identify the cleartext. Unfortunately, a standard notion of bisimulation would require that $P(M)$ and $P(N)$ send identical messages. So we should relax the definition of bisimulation to permit indistinguishable messages.

- In reasoning about a protocol, we need to consider its behaviour in reaction to inputs from the environment. These inputs are not entirely arbitrary. For example, consider a system $P(M)$ which discloses M when it receives a certain password. Assuming that the password remains secret, $P(M)$ should be equivalent to $P(N)$. In order to argue this, we need to characterise the set of possible inputs from the environment, and to show that it cannot include the password.
- Two messages that are indistinguishable at one point in time may become distinguishable later on. In particular, the keys under which they are encrypted may be disclosed to the environment, which may then inspect the cleartext that these keys were concealing. Thus, the notion of indistinguishability should be sensitive to future events.
  Conversely, the set of possible inputs from the environment grows with time, as the environment intercepts messages and learns values that were previously secret.

In short, a definition of bisimulation for cryptographic protocols should explain what outputs are indistinguishable for the environment, and what inputs the environment can generate at any point in time. In this paper we introduce a definition of bisimulation that provides the necessary account of the knowledge of the environment. As we show, bisimulation can be used for reasoning about examples like those sketched informally above. More generally, bisimulation can be used for proving authenticity and secrecy properties of protocols, and also for justifying certain protocol optimisations.

We develop our bisimulation proof technique in the context of the spi calculus [AG97a,AG97b,AG97c,Aba97], an extension of the pi calculus [MPW92] with cryptographic primitives. For simplicity, we consider only shared-key cryptography, although we believe that public-key cryptography could be treated through similar methods. Within the spi calculus, we prove the soundness of our technique. More precisely, we prove that bisimulation yields a sufficient condition for testing equivalence, the relation that we commonly use for specifications in the spi calculus.

We have developed other proof techniques for the spi calculus in earlier work. The one presented in this paper is a useful addition to our set of tools. Its distinguishing characteristic is its kinship to bisimulation proof techniques for other classes of systems. In particular, bisimulation proofs can often be done without creativity, essentially by state-space exploration.

The next section is a review of the spi calculus. Section 3 describes our proof method and Section 4 illustrates its use through some small examples. Section 5 discusses related work. Section 6 concludes.

2 The Spi Calculus (Review)

This section reviews the spi calculus, borrowing from earlier presentations. It gives the syntax and informal semantics of the spi calculus, introduces the main notations for its operational semantics, and finally defines testing equivalence.

2.1 Syntax

We assume an infinite set of *names* and an infinite set of *variables.* We let c, m, n, p, q, and r range over names, and let w, x, y, and z range over variables. When they represent keys, we let k and K range over names too.

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

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

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]. In examples, we write 1 as a shorthand for the term $suc(0)$.

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

$P, Q, R ::=$	processes
$\overline{M}\langle N\rangle.P$	output
$M(x).P$	input (scope of x is P)
$P \mid Q$	composition
$(\nu n)P$	restriction (scope of n is P)
$!P$	replication
$[M \textit{ is } N]\ P$	match
$\mathbf{0}$	nil
$\textit{let } (x, y) = M \textit{ in } P$	pair splitting (scope of x, y is P)
$\textit{case } M \textit{ of } 0 : P\ suc(x) : Q$	integer case (scope of x is Q)
$\textit{case } L \textit{ of } \{x\}_N \textit{ in } P$	decryption (scope of x is P)

We abbreviate $\overline{M}\langle N\rangle.\mathbf{0}$ to $\overline{M}\langle N\rangle$. 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. 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.
- 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 replicas 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 $let\ (x, y) = M\ in\ P$ behaves as $P[N/x][L/y]$ if M is a pair (N, L), and it is stuck if M is not a pair.
- An *integer case* process $case\ M\ of\ 0 : P\ suc(x) : Q$ behaves as P if M is 0, as $Q[N/x]$ if M is $suc(N)$ for some N, and otherwise is stuck.
- A *decryption* process $case\ L\ of\ \{x\}_N\ in\ P$ 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, $P \triangleq m(x).case\ x\ of\ \{y\}_K\ in\ \overline{m}\langle\{0\}_y\rangle$ is a process that is ready to receive a message x on the channel m. When the message is a ciphertext of the form $\{y\}_K$, process P sends 0 encrypted under y on the channel m. This process may be put in parallel with a process $Q \triangleq \overline{m}\langle\{K'\}_K\rangle$, which sends the name K' encrypted under K on the channel m. In order to restrict the use of K to P and Q, we may form $(\nu K)(P \mid Q)$. The environment of $(\nu K)(P \mid Q)$ will not be able to construct any message of the form $\{y\}_K$, since K is bound. Therefore, the component P of $(\nu K)(P \mid Q)$ may output $\{0\}_{K'}$, but not $\{0\}_z$ for any z different from K'. Alternatively, the component P of $(\nu K)(P \mid Q)$ may produce no output: for example, were it to receive 0 on the channel m, it would get stuck.

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

2.2 Operational Semantics

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. A *concretion* is an expression of the form $(\nu m_1, \dots, m_k)\langle M\rangle P$, where M is a term, P is a process, $k \geq 0$, and the names $m_1, \dots, m_k$ are bound in M and P. Intuitively, $(\nu m_1, \dots, m_k)\langle M\rangle P$ is like the process $(\nu m_1) \dots (\nu m_k)\overline{p}\langle M\rangle P$ minus the name p, provided p is not one of $m_1, \dots, m_k$. We often write concretions as $(\nu\vec{m})\langle M\rangle P$, where $\vec{m} = m_1, \dots, m_k$, 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 $fv(A)$ and $fn(A)$ be the sets of free variables and free names of an agent A, respectively.

A *barb* is a name m (representing input) or a co-name $\overline{m}$ (representing output). An *action* is a barb or the distinguished *silent action* τ. The *commitment relation* is written $P \xrightarrow{\alpha} A$, where P is a closed process, α is an action, and A is a closed agent. The exact definition of commitment appears in earlier papers on the spi calculus [AG97b,AG97c]; informally, the definition says:

- $P \xrightarrow{\tau} Q$ means that P becomes Q in one silent step (a τ step).
- $P \xrightarrow{m} (x)Q$ means that, in one step, P is ready to receive an input x on m and then to become Q.
- $P \xrightarrow{\overline{m}} (\nu m_1, \dots, m_k)\langle M\rangle Q$ means that, in one step, P is ready to create the new names $m_1, \dots, m_k$, to send M on m, and then to become Q.

2.3 Testing Equivalence

We say that two closed processes P and Q are testing equivalent, and write $P \simeq Q$, when for every closed process R and every barb β, if

$$P \mid R \xrightarrow{\tau}{}^{*} P' \xrightarrow{\beta} A$$

for some P' and A, then

$$Q \mid R \xrightarrow{\tau}{}^{*} Q' \xrightarrow{\beta} B$$

for some Q' and B, and vice versa.

For example, the processes $(\nu K)\overline{m}\langle\{0\}_K\rangle$ and $(\nu K)\overline{m}\langle\{1\}_K\rangle$ are testing equivalent. We may interpret this equivalence as a security property, namely that the process $(\nu K)\overline{m}\langle\{x\}_K\rangle$ does not reveal to its environment whether x is 0 or 1. In the examples contained in Section 4 and in earlier papers, various other properties (including, in particular, secrecy properties) are formulated in terms of testing equivalence.

In this paper we develop a sound technique for proving testing equivalence: we introduce a definition of bisimulation and show that if two closed processes are in one of our bisimulation relations then they are testing equivalent.

3 Framed Bisimulation

This section defines our notion of bisimulation, which we call *framed bisimulation.*

3.1 Frames and Theories

Our definition of bisimulation is based on the notions of a *frame* and of a *theory.* A bisimulation does not simply relate two processes P and Q, but instead relates two processes P and Q in the context of a frame and a theory. The frame and the theory represent the knowledge of the environment of P and Q.

- A frame is a finite set of names. Intuitively, a frame is a set of names available to the environment of the processes P and Q. We use fr to range over frames.
- A theory is a finite set of pairs of terms. Intuitively, a theory that includes a pair (M, N) indicates that the environment cannot distinguish the data M coming from process P and the data N coming from process Q. We use th to range over theories.

Next we define the predicate $(fr, th) \vdash M \leftrightarrow N$ inductively, by a set of rules. Intuitively, this predicate means that the environment cannot distinguish M coming from P and N coming from Q, and that the environment has (or can construct) M in interaction with P and N in interaction with Q.

(Eq Frame) $\quad$ (Eq Theory) $\quad$ (Eq Variable)

$$\frac{n \in fr}{(fr, th) \vdash n \leftrightarrow n} \qquad \frac{(M, N) \in th}{(fr, th) \vdash M \leftrightarrow N} \qquad \frac{}{(fr, th) \vdash x \leftrightarrow x}$$

(Eq Pair)
$$\frac{(\mathit{fr},\mathit{th}) \vdash M \leftrightarrow M' \quad (\mathit{fr},\mathit{th}) \vdash N \leftrightarrow N'}{(\mathit{fr},\mathit{th}) \vdash (M,N) \leftrightarrow (M',N')}$$

(Eq Zero)
$$\frac{}{(\mathit{fr},\mathit{th}) \vdash 0 \leftrightarrow 0}$$

(Eq Suc)
$$\frac{(\mathit{fr},\mathit{th}) \vdash M \leftrightarrow M'}{(\mathit{fr},\mathit{th}) \vdash \mathit{suc}(M) \leftrightarrow \mathit{suc}(M')}$$

(Eq Encrypt)
$$\frac{(\mathit{fr},\mathit{th}) \vdash M \leftrightarrow M' \quad (\mathit{fr},\mathit{th}) \vdash N \leftrightarrow N'}{(\mathit{fr},\mathit{th}) \vdash \{M\}_N \leftrightarrow \{M'\}_{N'}}$$

For example, if $\mathit{fr} = \{n\}$ and $\mathit{th} = \{(\{0\}_K, \{n\}_K)\}$, where n and K are distinct names, then we have $(\mathit{fr},\mathit{th}) \vdash n \leftrightarrow n$ and $(\mathit{fr},\mathit{th}) \vdash \{0\}_K \leftrightarrow \{n\}_K$, and also $(\mathit{fr},\mathit{th}) \vdash (n, \{0\}_K) \leftrightarrow (n, \{n\}_K)$, but we have neither $(\mathit{fr},\mathit{th}) \vdash K \leftrightarrow K$ nor $(\mathit{fr},\mathit{th}) \vdash \{n\}_K \leftrightarrow \{0\}_K$.

We say that the pair $(\mathit{fr},\mathit{th})$ is ok, and write $(\mathit{fr},\mathit{th}) \vdash \mathit{ok}$, if two conditions hold:

(1) whenever $(M,N) \in \mathit{th}$:
   - M is closed and there are terms M_1 and M_2 such that $M = \{M_1\}_{M_2}$ and there is no N_2 such that $(\mathit{fr},\mathit{th}) \vdash M_2 \leftrightarrow N_2$;
   - N is closed and there are terms N_1 and N_2 such that $N = \{N_1\}_{N_2}$ and there is no M_2 such that $(\mathit{fr},\mathit{th}) \vdash M_2 \leftrightarrow N_2$;

(2) whenever $(M,N) \in \mathit{th}$ and $(M',N') \in \mathit{th}$, $M = M'$ if and only if $N = N'$.

Intuitively, the first condition requires that each term in a pair (M,N) in a theory be formed by ciphertexts that the environment cannot decrypt. For example, the requirement that there be no N_2 such that $(\mathit{fr},\mathit{th}) \vdash M_2 \leftrightarrow N_2$ means that the environment cannot construct M_2 for decrypting M. The second condition guarantees that no ciphertext is equated to two other ciphertexts. This condition is essential because the environment can compare ciphertexts even when it cannot decrypt them (see Example 2 of Section 4).

3.2 Ordering Frame-Theory Pairs

We define an ordering between pairs of frames and theories as follows: we let $(\mathit{fr},\mathit{th}) \leq (\mathit{fr}',\mathit{th}')$ if and only if for all M and N, $(\mathit{fr},\mathit{th}) \vdash M \leftrightarrow N$ implies $(\mathit{fr}',\mathit{th}') \vdash M \leftrightarrow N$. This relation is reflexive and transitive. It is not the same as the pairwise ordering induced by subset inclusion; $\mathit{fr} \subseteq \mathit{fr}'$ and $\mathit{th} \subseteq \mathit{th}'$ imply $(\mathit{fr},\mathit{th}) \leq (\mathit{fr}',\mathit{th}')$, but the converse implication does not hold.

Proposition 1. *Suppose $(\mathit{fr}',\mathit{th}') \vdash \mathit{ok}$. Then $(\mathit{fr},\mathit{th}) \leq (\mathit{fr}',\mathit{th}')$ if and only if $\mathit{fr} \subseteq \mathit{fr}'$ and $(\mathit{fr}',\mathit{th}') \vdash M \leftrightarrow N$ for each $(M,N) \in \mathit{th}$.*

As indicated above, we view a pair $(\mathit{fr},\mathit{th})$ as a representation for the knowledge of an environment. With this view, and assuming that $(\mathit{fr}',\mathit{th}') \vdash \mathit{ok}$, the relation $(\mathit{fr},\mathit{th}) \leq (\mathit{fr}',\mathit{th}')$ means that the environment may go from the knowledge represented in $(\mathit{fr},\mathit{th})$ to the knowledge represented in $(\mathit{fr}',\mathit{th}')$. The definition of $(\mathit{fr},\mathit{th}) \leq (\mathit{fr}',\mathit{th}')$ implies that the set of names and terms that the environment has (or can construct) grows in this transition. It also implies that any indistinguishable pair of terms remains indistinguishable after the transition. So, if ever we assert that (fr, th) characterises an environment, we should take care that (fr, th) does not imply that the environment cannot distinguish two terms M and N if later information would allow the environment to distinguish these terms. For example, if fr' includes the name n, then th should not contain $(\{0\}_n, \{1\}_n)$. Intuitively, the acquisition of the name n would allow the environment to distinguish $\{0\}_n$ and $\{1\}_n$, so $(fr', th') \vdash \{0\}_n \leftrightarrow \{1\}_n$ would not hold. On the other hand, th may contain $(\{0\}_n, \{0\}_n)$; in that case, th' could not contain $(\{0\}_n, \{0\}_n)$, but we would at least have $(fr', th') \vdash \{0\}_n \leftrightarrow \{0\}_n$.

3.3 Framed Relations and Bisimulations

For a theory th, we let $fn(th) = \bigcup\{fn(M) \cup fn(N) \mid (M, N) \in th\}$. We let $\pi_1(th) = \{M \mid (M, N) \in th\}$ and $\pi_2(th) = \{N \mid (M, N) \in th\}$, and write $fn(\pi_1(th))$ and $fn(\pi_2(th))$ for the sets of names $\bigcup\{fn(M) \mid M \in \pi_1(th)\}$ and $\bigcup\{fn(N) \mid N \in \pi_2(th)\}$ respectively.

A *framed process pair* is a quadruple (fr, th, P, Q) such that P and Q are closed processes, fr is a frame, and th is a theory. When $\mathcal{R}$ is a set of framed process pairs, we write $(fr, th) \vdash P \mathrel{\mathcal{R}} Q$ to mean $(fr, th, P, Q) \in \mathcal{R}$. A *framed relation* is a set $\mathcal{R}$ of framed process pairs such that $(fr, th) \vdash ok$ whenever $(fr, th) \vdash P \mathrel{\mathcal{R}} Q$.

A *framed simulation* is a framed relation $\mathcal{S}$ such that, whenever $(fr, th) \vdash P \mathrel{\mathcal{S}} Q$, the following three conditions hold.

- If $P \xrightarrow{\tau} P'$ then there is a process Q' with $Q \xrightarrow{\tau} Q'$ and $(fr, th) \vdash P' \mathrel{\mathcal{S}} Q'$.
- If $P \xrightarrow{c} (x)P'$ and $c \in fr$ then there is an abstraction $(x)Q'$ with $Q \xrightarrow{c} (x)Q'$ and, for all sets $\{\vec{n}\}$ disjoint from $fn(P) \cup fn(Q) \cup fr \cup fn(th)$ and all closed M and N, if $(fr \cup \{\vec{n}\}, th) \vdash M \leftrightarrow N$ then $(fr \cup \{\vec{n}\}, th) \vdash P'[M/x] \mathrel{\mathcal{S}} Q'[N/x]$.
- If $P \xrightarrow{\bar{c}} (\nu\vec{m})\langle M\rangle P'$, $c \in fr$, and the set $\{\vec{m}\}$ is disjoint from $fn(P) \cup fn(\pi_1(th)) \cup fr$ then there is a concretion $(\nu\vec{n})\langle N\rangle Q'$ with $Q \xrightarrow{\bar{c}} (\nu\vec{n})\langle N\rangle Q'$ and the set $\{\vec{n}\}$ is disjoint from $fn(Q) \cup fn(\pi_2(th)) \cup fr$, and there is a frame-theory pair (fr', th') such that $(fr, th) \leq (fr', th')$, $(fr', th') \vdash M \leftrightarrow N$, and $(fr', th') \vdash P' \mathrel{\mathcal{S}} Q'$.

We may explain these conditions as follows.

- The first condition simply requires that if P can take a τ step then Q can match this step.
- The second condition concerns input steps where the channel c on which the input happens is in fr (that is, it is known to the environment). In this case, we must consider the possible inputs M from the environment to $(x)P'$, namely the terms M that the environment can construct according to $(fr \cup \{\vec{n}\}, th)$. The names in $\vec{n}$ are fresh names, intuitively names just generated by the environment. Correspondingly, we consider the possible inputs N

for $(x)Q'$, for an appropriate $(x)Q'$ obtained from Q. We then require that giving these inputs to $(x)P'$ and $(x)Q'$, respectively, yields related processes $P'[M/x]$ and $Q'[N/x]$.
The choice of $(x)Q'$ is independent of the choices of M and N. So, in the technical jargon, we may say that $\mathcal{S}$ is a *late* framed simulation.

- The third condition concerns output steps where the channel c on which the output happens is in fr (that is, it is known to the environment). In this case, P outputs the term M while creating the names $\vec{m}$. The condition requires that Q can output a corresponding term N while creating some names $\vec{n}$. It also constrains M and N, and the resulting processes P' and Q'. The constraints concern a new frame-theory pair (fr', th'). Intuitively, this pair represents the knowledge of the environment after the output step. The requirement that $(fr', th') \vdash M \leftrightarrow N$ means that the environment obtains M in interaction with P and N in interaction with Q, and that it should not be able to distinguish them from one another.
  Because we do not impose a minimality requirement on (fr', th'), this pair may attribute "too much" knowledge to the environment. For example, fr' may contain names that are neither in fr nor in M or N, so intuitively the environment would not be expected to know these names. On the other hand, the omission of a minimality requirement results in simpler definitions, and does not compromise soundness.

A *framed bisimulation* is a framed relation $\mathcal{S}$ such that both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are framed simulations. *Framed bisimilarity* (written $\sim_f$) is the greatest framed bisimulation. By the Knaster-Tarski fixpoint theorem, since the set of framed relations ordered by subset inclusion forms a complete lattice, framed bisimilarity exists, and equals the union of all framed bisimulations.

Our intent is that our definition of framed bisimulation may serve as the basis for an algorithm, at least for finite-state processes. Unfortunately, the definition contains several levels of quantification. The universal quantifiers present a serious obstacle to any algorithm for constructing framed bisimulations. In particular, the condition for input steps concerns all possible inputs M and N; these inputs are of unbounded size, and may contain an arbitrary number of fresh names. However, we conjecture that the inputs can be classified according to a finite number of patterns—intuitively, because the behaviour of any finite-state process can depend on at most a finite portion of its inputs. An algorithm for constructing framed bisimulations might consider all inputs that match the same pattern at once. We leave the invention of such an algorithm for future work.

3.4 Soundness (Summary)

Our main soundness theorem about framed bisimulation is that it is a sufficient condition for testing equivalence.

Theorem 2. *Consider any closed processes P and Q, and any name $n \notin fn(P) \cup fn(Q)$. Suppose that $(fn(P) \cup fn(Q) \cup \{n\}, \emptyset) \vdash P \sim_f Q$. Then $P \simeq Q$.*

This theorem implies that if we want to prove that two processes are testing equivalent, then we may construct a framed bisimulation $\mathcal{S}$ such that $(\mathit{fn}(P) \cup \mathit{fn}(Q) \cup \{n\}, \emptyset) \vdash P \mathrel{\mathcal{S}} Q$ where n is a single, arbitrary new name. (The addition of the name n is technically convenient, but may not be necessary.) The next section illustrates this approach through several examples.

The proof of this theorem requires a number of auxiliary notations, definitions, and lemmas. We omit the details of the proof, and only indicate its main idea. In the course of the proof, we extend the relation $\leftrightarrow$ to processes: we define the predicate $(\mathit{fr}, \mathit{th}) \vdash P \leftrightarrow Q$ by the following rules.

(Eq Out)

$$\frac{(\mathit{fr}, \mathit{th}) \vdash M \leftrightarrow M' \quad (\mathit{fr}, \mathit{th}) \vdash N \leftrightarrow N' \quad (\mathit{fr}, \mathit{th}) \vdash P \leftrightarrow P'}{(\mathit{fr}, \mathit{th}) \vdash \overline{M}\langle N\rangle.P \leftrightarrow \overline{M'}\langle N'\rangle.P'}$$

(Eq In)

$$\frac{(\mathit{fr}, \mathit{th}) \vdash M \leftrightarrow M' \quad (\mathit{fr}, \mathit{th}) \vdash P \leftrightarrow P'}{(\mathit{fr}, \mathit{th}) \vdash M(x).P \leftrightarrow M'(x).P'}$$

(Eq Repl)

$$\frac{(\mathit{fr}, \mathit{th}) \vdash P \leftrightarrow P'}{(\mathit{fr}, \mathit{th}) \vdash !P \leftrightarrow !P'}$$

(Eq Par)

$$\frac{(\mathit{fr}, \mathit{th}) \vdash P \leftrightarrow P' \quad (\mathit{fr}, \mathit{th}) \vdash Q \leftrightarrow Q'}{(\mathit{fr}, \mathit{th}) \vdash P \mid Q \leftrightarrow P' \mid Q'}$$

(Eq Res) (where $n \notin \mathit{fr} \cup \mathit{fn}(\mathit{th})$)

$$\frac{(\mathit{fr} \cup \{n\}, \mathit{th}) \vdash P \leftrightarrow P'}{(\mathit{fr}, \mathit{th}) \vdash (\nu n)P \leftrightarrow (\nu n)P'}$$

(Eq Match)

$$\frac{(\mathit{fr}, \mathit{th}) \vdash M \leftrightarrow M' \quad (\mathit{fr}, \mathit{th}) \vdash N \leftrightarrow N' \quad (\mathit{fr}, \mathit{th}) \vdash P \leftrightarrow P'}{(\mathit{fr}, \mathit{th}) \vdash [M \ \mathit{is}\ N]\, P \leftrightarrow [M' \ \mathit{is}\ N']\, P'}$$

(Eq Nil)

$$\frac{}{(\mathit{fr}, \mathit{th}) \vdash \mathbf{0} \leftrightarrow \mathbf{0}}$$

(Eq Let)

$$\frac{(\mathit{fr}, \mathit{th}) \vdash M \leftrightarrow M' \quad (\mathit{fr}, \mathit{th}) \vdash P \leftrightarrow P'}{(\mathit{fr}, \mathit{th}) \vdash \mathit{let}\ (x, y) = M \ \mathit{in}\ P \leftrightarrow \mathit{let}\ (x, y) = M' \ \mathit{in}\ P'}$$

(Eq IntCase)

$$\frac{(\mathit{fr}, \mathit{th}) \vdash M \leftrightarrow M' \quad (\mathit{fr}, \mathit{th}) \vdash P \leftrightarrow P' \quad (\mathit{fr}, \mathit{th}) \vdash Q \leftrightarrow Q'}{(\mathit{fr}, \mathit{th}) \vdash \mathit{case}\ M \ \mathit{of}\ 0 : P \ \mathit{suc}(x) : Q \leftrightarrow \mathit{case}\ M' \ \mathit{of}\ 0 : P' \ \mathit{suc}(x) : Q'}$$

(Eq Decrypt)

$$\frac{(\mathit{fr}, \mathit{th}) \vdash M \leftrightarrow M' \quad (\mathit{fr}, \mathit{th}) \vdash N \leftrightarrow N' \quad (\mathit{fr}, \mathit{th}) \vdash P \leftrightarrow P'}{(\mathit{fr}, \mathit{th}) \vdash \mathit{case}\ M \ \mathit{of}\ \{x\}_N \ \mathit{in}\ P \leftrightarrow \mathit{case}\ M' \ \mathit{of}\ \{x\}_{N'} \ \mathit{in}\ P'}$$

The core of the proof depends on a relation, $\mathcal{S}$, defined so that $P \mathrel{\mathcal{S}} Q$ if and only if there is a frame fr, a theory th, and processes P_1, P_2, Q_1, Q_2, such that

$$P = (\nu \vec{p})(P_1 \mid P_2) \qquad Q = (\nu \vec{q})(Q_1 \mid Q_2)$$

and $(\mathit{fr}, \mathit{th}) \vdash \mathit{ok}$, $(\mathit{fr}, \mathit{th}) \vdash P_1 \sim_f Q_1$, and $(\mathit{fr}, \mathit{th}) \vdash P_2 \leftrightarrow Q_2$, where $\{\vec{p}\} = (\mathit{fn}(P_1) \cup \mathit{fn}(\pi_1(\mathit{th}))) - \mathit{fr}$ and $\{\vec{q}\} = (\mathit{fn}(Q_1) \cup \mathit{fn}(\pi_2(\mathit{th}))) - \mathit{fr}$. By a detailed case analysis, we may show that $\mathcal{S}$ satisfies the definition of a standard notion of bisimulation—a barbed bisimulation up to restriction and barbed equivalence. Given some auxiliary lemmas about testing equivalence, the theorem then follows easily. The construction of $\mathcal{S}$ also yields that framed bisimilarity is a sufficient condition for a strong equivalence called barbed congruence.

The converse of soundness—completeness—does not hold. The failure of completeness follows from the fact that framed bisimilarity is a sufficient condition for barbed congruence. (Barbed congruence and a fortiori framed bisimilarity are sensitive to τ steps and to branching structure, while testing equivalence is not.) Incompleteness may be somewhat unfortunate but, in our experience, it seems to be compatible with usefulness.

4 Examples

This section shows how bisimulations can be exploited in proofs through some small examples. These examples could not be handled by standard notions of bisimulation (like that of our earlier work [AG97b,AG97c]). We have worked through further examples, including some examples with more steps. In all cases, the proofs are rather straightforward.

Throughout this section, c, K, K_1, and K_2 are distinct names, and n is any name different from c. Moreover, M, M', M'', M_1, M_2, N_1, and N_2 are closed terms; it is convenient to assume that no name occurs in them.

Example 1 As a first example, we show that the processes $(\nu K)\overline{c}\langle\{M\}_K\rangle$ and $(\nu K)\overline{c}\langle\{M'\}_K\rangle$ are in a framed bisimulation, so they are testing equivalent. Intuitively, this means that these processes do not reveal M and M', respectively.

For this example, we let $\mathcal{S}$ be the least relation such that:

- $(\{c,n\},\emptyset) \vdash (\nu K)\overline{c}\langle\{M\}_K\rangle \;\mathcal{S}\; (\nu K)\overline{c}\langle\{M'\}_K\rangle$
- $(\{c,n\},\{(\{M\}_k,\{M'\}_k)\}) \vdash \mathbf{0} \;\mathcal{S}\; \mathbf{0}$
  for all names $k \notin \{c,n\}$

Since $(\{c,n\},\emptyset) \vdash ok$ and $(\{c,n\},\{(\{M\}_k,\{M'\}_k)\}) \vdash ok$, $\mathcal{S}$ is a framed relation. Next we show that $\mathcal{S}$ is a framed simulation; symmetric reasoning establishes that $\mathcal{S}^{-1}$ is one too. Assuming that $(fr,th) \vdash P \;\mathcal{S}\; Q$, we need to examine the commitments of P and Q. We consider two cases, which correspond to the two clauses of the definition of $\mathcal{S}$.

- Suppose that $P = (\nu K)\overline{c}\langle\{M\}_K\rangle$ and $Q = (\nu K)\overline{c}\langle\{M'\}_K\rangle$. In this case, we have $fr = \{c,n\}$ and $th = \emptyset$. Up to renaming of the bound name K, the only commitment of P is $P \xrightarrow{\overline{c}} (\nu K)\langle\{M\}_K\rangle\mathbf{0}$. To establish that $\mathcal{S}$ is a framed simulation, we need only consider the case where K is renamed to some $k \notin fn(P) \cup fn(\pi_1(\emptyset)) \cup \{c,n\}$, that is, $k \notin \{c,n\}$. By renaming, we have $Q \xrightarrow{\overline{c}} (\nu k)\langle\{M'\}_k\rangle\mathbf{0}$. We let $th' = \{(\{M\}_k,\{M'\}_k)\}$. We have $(fr,th) \leq (fr,th')$, $(fr,th') \vdash \{M\}_k \leftrightarrow \{M'\}_k$, and $(fr,th') \vdash \mathbf{0} \leftrightarrow \mathbf{0}$. Thus, Q can match P's commitment.

- Suppose that $P = \mathbf{0}$ and $Q = \mathbf{0}$. This case is trivial, since $\mathbf{0}$ has no commitments.

In short, $(\{c,n\},\emptyset) \vdash (\nu K)\overline{c}\langle\{M\}_K\rangle \; \mathcal{S} \; (\nu K)\overline{c}\langle\{M'\}_K\rangle$, and $\mathcal{S}$ is a framed bisimulation, as desired.

Example 2 As a small variant of the first example, we consider the processes $(\nu K)\overline{c}\langle(\{M\}_K,\{M\}_K)\rangle$ and $(\nu K)\overline{c}\langle(\{M'\}_K,\{M''\}_K)\rangle$.

When $M' = M''$, the argument of the first example works for this example too, with only trivial modifications. We define $\mathcal{S}$ as the least relation such that:

- $(\{c,n\},\emptyset) \vdash (\nu K)\overline{c}\langle(\{M\}_K,\{M\}_K)\rangle \; \mathcal{S} \; (\nu K)\overline{c}\langle(\{M'\}_K,\{M''\}_K)\rangle$
- $(\{c,n\},\{(\{M\}_k,\{M'\}_k),(\{M\}_k,\{M''\}_k)\}) \vdash \mathbf{0} \; \mathcal{S} \; \mathbf{0}$
  for all names $k \notin \{c,n\}$

This relation is a framed bisimulation when $M' = M''$. On the other hand, it is not a framed bisimulation when $M' \neq M''$. In fact, in that case it is not even a framed relation, because $(\{c,n\},\{(\{M\}_k,\{M'\}_k),(\{M\}_k,\{M''\}_k)\}) \vdash ok$ does not hold (because condition (2) of the definition of ok is not satisfied).

The fact that $\mathcal{S}$ is not a framed bisimulation in this case should not be a concern. It is actually necessary: the processes $(\nu K)\overline{c}\langle(\{M\}_K,\{M\}_K)\rangle$ and $(\nu K)\overline{c}\langle(\{M'\}_K,\{M''\}_K)\rangle$ are not testing equivalent when $M' \neq M''$. The environment $c(z).let\ (x,y) = z\ in\ [x\ is\ y]\ \overline{c}\langle 0\rangle$ distinguishes them. Thus, this example illustrates that two ciphertexts that cannot be decrypted can still be compared, and justifies part of the definition of framed bisimulation.

Example 3 As a further variant, we study an example with nested encryption. We consider the processes $(\nu K_1)(\nu K_2)\overline{c}\langle\{M_1,\{M_2\}_{K_2}\}_{K_1}\rangle.\overline{c}\langle K_1\rangle$ and $(\nu K_1)(\nu K_2)$ $\overline{c}\langle\{N_1,\{N_2\}_{K_2}\}_{K_1}\rangle.\overline{c}\langle K_1\rangle$. Each of these processes creates two keys K_1 and K_2, sends a ciphertext, and then reveals K_1. Anyone who receives K_1 can partially decrypt the ciphertext.

In order to analyse these processes, we let $\mathcal{S}$ be the least relation such that:

- $(\{c,n\},\emptyset) \vdash (\nu K_1)(\nu K_2)\overline{c}\langle\{M_1,\{M_2\}_{K_2}\}_{K_1}\rangle.\overline{c}\langle K_1\rangle \; \mathcal{S}$
  $(\nu K_1)(\nu K_2)\overline{c}\langle\{N_1,\{N_2\}_{K_2}\}_{K_1}\rangle.\overline{c}\langle K_1\rangle$
- $(\{c,n,k_1\},\{(\{M_2\}_{k_2},\{N_2\}_{k_2})\}) \vdash \overline{c}\langle k_1\rangle \; \mathcal{S} \; \overline{c}\langle k_1\rangle$
  for all names k_1, k_2 with $k_1 \neq k_2$ and $\{k_1,k_2\} \cap \{c,n\} = \emptyset$
- $(\{c,n,k_1\},\{(\{M_2\}_{k_2},\{N_2\}_{k_2})\}) \vdash \mathbf{0} \; \mathcal{S} \; \mathbf{0}$
  for all names k_1, k_2 with $k_1 \neq k_2$ and $\{k_1,k_2\} \cap \{c,n\} = \emptyset$

Note how, between the first and the second clauses, the frame has been enlarged with k_1, although the processes considered in the second clause have not yet sent k_1; this simplifies the construction of $\mathcal{S}$ and is permitted by the definitions of Section 3. The assumptions guarantee that $(\{c,n,k_1\},\{(\{M_2\}_{k_2},\{N_2\}_{k_2})\}) \vdash ok$ and hence that $\mathcal{S}$ is a framed relation. Moreover, $\mathcal{S}$ is a framed bisimulation if and only if the following condition holds:

$$(\{c,n,k_1\},\{(\{M_2\}_{k_2},\{N_2\}_{k_2})\}) \vdash \{M_1,\{M_2\}_{k_2}\}_{k_1} \leftrightarrow \{N_1,\{N_2\}_{k_2}\}_{k_1}$$

In turn, this condition holds if and only if $M_1 = N_1$. Intuitively, the equality $M_1 = N_1$ becomes necessary only when the two processes send the key k_1, since M_1 and N_1 are not visible in the first message. Our definition of $\leq$ guarantees that the necessity of $M_1 = N_1$ is propagated correctly.

Example 4 While all the examples above concern the secrecy of certain outputs, this one concerns the impossibility of certain inputs. We consider the processes $(\nu K)\overline{c}\langle\{0\}_K\rangle.c(x).[x \textit{ is } K]\,\overline{c}\langle\{0\}_K\rangle$ and $(\nu K)\overline{c}\langle\{0\}_K\rangle.c(x).\mathbf{0}$. The former process creates a key K, sends $\{0\}_K$, listens for an input, and if it receives K then it sends $\{0\}_K$ again. However, we would expect that K will never arrive as an input to this process, since the process never discloses K (but only $\{0\}_K$, from which K itself cannot be deduced). Therefore, we would expect this process to be equivalent to the latter process, which simply stops upon receipt of a message.

For this example, we let $\mathcal{S}$ be the least relation such that:

- $(\{c,n\},\emptyset) \vdash ((\nu K)\overline{c}\langle\{0\}_K\rangle.c(x).[x \textit{ is } K]\,\overline{c}\langle\{0\}_K\rangle)\ \mathcal{S}\ ((\nu K)\overline{c}\langle\{0\}_K\rangle.c(x).\mathbf{0})$
- $(\{c,n\},\{(\{0\}_k,\{0\}_k)\}) \vdash (c(x).[x \textit{ is } k]\,\overline{c}\langle\{0\}_k\rangle)\ \mathcal{S}\ (c(x).\mathbf{0})$
  for all names k with $k \notin \{c,n\}$
- $(\{c,n,\vec{m}\},\{(\{0\}_k,\{0\}_k)\}) \vdash ([N \textit{ is } k]\,\overline{c}\langle\{0\}_k\rangle)\ \mathcal{S}\ \mathbf{0}$
  for all names k with $k \notin \{c,n\}$, for all sets $\{\vec{m}\}$ disjoint from $\{c,n,k\}$, and all closed terms N and N' with $(\{c,n,\vec{m}\},\{(\{0\}_k,\{0\}_k)\}) \vdash N \leftrightarrow N'$
  (We are not assuming that no names occur in the closed terms N and N'.)

Since $(\{c,n,\vec{m}\},\{(\{0\}_k,\{0\}_k)\}) \vdash N \leftrightarrow N'$ and $k \notin \{c,n,\vec{m}\}$, the term N is not k, so $[N \textit{ is } k]\ \overline{c}\langle\{0\}_k\rangle \xrightarrow{\alpha} A$ is not true for any α and A. In other words, the process $[N \textit{ is } k]\,\overline{c}\langle\{0\}_k\rangle$ is stuck. It follows easily that $\mathcal{S}$ is a framed bisimulation.

Example 5 In cryptographic protocols, some keys are generated by consulting sources of randomness, but it is also common to generate keys by applying one-way functions to other keys. (Whereas many one-way functions are quite efficient, randomness and key agreement can be relatively expensive [Sch96b].) As a final example, we consider a simple protocol transformation inspired by a common method for generating keys. We compare the process $(\nu K_1)(\nu K_2)$ $\overline{c}\langle\{M_1\}_{K_1}\rangle.\overline{c}\langle\{M_2\}_{K_2}\rangle$, which generates and uses two keys, with the process (νK) $\overline{c}\langle\{N_1\}_{\{0\}_K}\rangle.\overline{c}\langle\{N_2\}_{\{1\}_K}\rangle$, which generates the master key K and then uses the derived keys $\{0\}_K$ and $\{1\}_K$.

In order to show that these processes are testing equivalent, we construct once more a framed bisimulation. We let $\mathcal{S}$ be the least relation such that:

- $(\{c,n\},\emptyset) \vdash (\nu K_1)(\nu K_2)\overline{c}\langle\{M_1\}_{K_1}\rangle.\overline{c}\langle\{M_2\}_{K_2}\rangle\ \mathcal{S}$
  $(\nu K)\overline{c}\langle\{N_1\}_{\{0\}_K}\rangle.\overline{c}\langle\{N_2\}_{\{1\}_K}\rangle$
- $(\{c,n\},\{(\{M_1\}_{k_1},\{N_1\}_{\{0\}_k})\}) \vdash (\nu K_2)\overline{c}\langle\{M_2\}_{K_2}\rangle\ \mathcal{S}\ \overline{c}\langle\{N_2\}_{\{1\}_k}\rangle$
  for all names k and k_1 with $\{k,k_1\} \cap \{c,n\} = \emptyset$
- $(\{c,n\},\{(\{M_1\}_{k_1},\{N_1\}_{\{0\}_k}),(\{M_2\}_{k_2},\{N_2\}_{\{1\}_k})\}) \vdash \mathbf{0}\ \mathcal{S}\ \mathbf{0}$
  for all names k and k_1 with $\{k,k_1\} \cap \{c,n\} = \emptyset$
  and all names k_2 with $k_2 \notin \{c,n,k_1\}$

It is somewhat laborious but not difficult to check that $\mathcal{S}$ is a framed bisimulation, much as in the examples above.

5 Related Work

Park [Par81] first suggested the bisimulation proof technique, in the context of Milner's CCS. After Park's work, bisimulation became a cornerstone of the theory of CCS [Mil89]. Milner, Parrow, and Walker [MPW92] extensively studied a variety of forms of bisimulation for the pi calculus, their generalisation of CCS with name-passing and mobile restrictions. Our definition of framed bisimulation generalises (and relaxes) the definition of strong bisimulation from earlier work on the spi calculus [AG97b,AG97c]. We can show that if processes P and Q are strongly bisimilar, then, for all frames fr, $(\mathit{fr}, \emptyset) \vdash P \sim_f Q$. The converse implication fails.

According to most other definitions, a bisimulation is a set of pairs of processes. According to our definition, a framed bisimulation is a set of quadruples consisting of a pair of processes grouped with a frame and a theory. According to a definition of Pierce and Sangiorgi [PS96] for a typed pi calculus, a bisimulation is a set of pairs of processes indexed by a type assumption that binds types to channel names. Our use of a frame is a little like their use of typing assumptions, in that both a frame and a typing assumption delimit the channels on which the environment may observe the processes in the bisimulation. On the other hand, the spi calculus is untyped, and our use of a theory to represent compound terms possessed but not decomposable by the environment seems to be new.

There are also parallels with the work of Pitts and Stark [PS93] on the ν-calculus, a simply-typed λ-calculus enriched with dynamic allocation of names. Pitts and Stark define a logical relation on programs, parameterised by a partial bijection on the names free in related programs. Their logical relation is sound for proving observational equivalence; it is incomplete but more generous than the usual notion of applicative bisimilarity for the ν-calculus. A logical relation is one in which two abstractions are related if and only if they send related arguments to related results. Given the clause for inputs in the definition of framed bisimulation, which requires the bodies of two abstractions $(x)P'$ and $(x)Q'$ to be related on all related terms M and N, we may say that the relations $(\mathit{fr}, \mathit{th}) \vdash M \leftrightarrow N$ and $(\mathit{fr}, \mathit{th}) \vdash P \sim_f Q$ form a parametric logical relation on the terms and processes of the spi calculus. Like Pitts and Stark's logical relation, our logical relation is sound for proving testing equivalence. Further, it is incomplete but more generous than the usual notion of strong bisimilarity; it has parameters (the frame and the theory) that serve to identify certain processes that are distinguished by the usual relation of strong bisimilarity. However, the analogy with Pitts and Stark's work is not perfect; in particular, their use of partial bijections on names is different from our use of frames and theories.

In the last few years, several methods for analysing cryptographic protocols have been developed within action-based or state-based models (see for example [MCF87,Mil95,Kem89,Mea92,GM95,Low96,Sch96a,Bol96,Pau97]). Some of these models are presented as process algebras, others in logical forms. Often, the analysis of a protocol requires defining a particular attacker (an environment) for the protocol; recently, there has been promising progress towards automating the construction of this attacker. Bisimulation techniques do appear in the security literature (as in the work of Focardi and Gorrieri [FG95]), but rarely, and without special tailoring to cryptographic applications.

6 Conclusions

When reasoning about a cryptographic protocol, we must take into account the knowledge of the environment with which the protocol interacts. In our definition of bisimulation, this knowledge is represented precisely as a set of names that the environment has obtained, and as a set of pairs of ciphertexts that the environment has received but cannot distinguish. This precise representation of the knowledge of the environment is the basis for an effective and sound proof technique. Using this technique, we can construct proofs for small but subtle cryptographic protocols. The proofs are fairly concise and do not require much creativity. Therefore, although we have not yet attempted to mechanise our proofs, we believe that such a mechanisation is possible, and that it may enable the automatic verification of substantial examples.

Acknowledgements

Discussions with Davide Sangiorgi were helpful at the start of this work. Andy Pitts made useful comments.

Gordon held a Royal Society University Research Fellowship at the University of Cambridge Computer Laboratory for most of the time we worked on this paper. He now holds a position at Microsoft Research.

References

[Aba97] M. Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 611–638. Springer-Verlag, 1997.

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

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

[AG97c] M. Abadi and A. D. Gordon. Reasoning about cryptographic protocols in the spi calculus. In *CONCUR'97: Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 59–73. Springer-Verlag, 1997.

[Bol96] D. Bolignano. An approach to the formal verification of cryptographic protocols. In *3rd ACM Conference on Computer and Communications Security*, pages 106–118, March 1996.

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

[FG95] R. Focardi and R. Gorrieri. A classification of security properties. *Journal of Computer Security*, 3(1), 1995.

[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.

[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.

[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.

[Par81] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical Computer Science: 5th GI-Conference, Karlsruhe*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, March 1981.

[Pau97] L. Paulson. Proving properties of security protocols by induction. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 70–83, 1997.

[PS93] A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *Mathematical Foundations of Computer Science, Proc. 18th Int. Symp., Gdańsk, 1993*, volume 711 of *Lecture Notes in Computer Science*, pages 122–141. Springer-Verlag, 1993.

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

[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.