Authentication primitives for secure protocol specifications

C. Bodei a,∗, P. Degano a, R. Focardi b, C. Priami c

a Dipartimento di Informatica, Università di Pisa, Via Buonarroti 2, I-56125 Pisa, Italy
b Dipartimento di Informatica, Università Ca’ Foscari di Venezia, Via Torino 155, I-30173 Venezia, Italy
c Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Via Sommarive 14, I-38050 Povo (TN), Italy

Available online 25 June 2004

Abstract

We use two authentication primitives proposed recently as a linguistic support for enforcing authentication. They offer a way of abstracting from various specifications of authentication and of obtaining idealized protocols “secure by construction”. Consequently, they help in proving that a cryptographic protocol correctly implements its corresponding abstract version; when the implementation is incorrect, suggestions on how to fix it may come from reasoning on the abstract specification.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Cryptographic process calculi; Security; Authentication

1. Introduction

In computer science, few concepts are at the same time so intuitively clear and so hopelessly elusive as that of security. It has to do with confidentiality, integrity and availability, but also with non-repudiation, authenticity and even more, depending on the current applications. The technology of distributed and parallel systems and networks deeply influences this topic: new problems and scenarios arise, making harder some of the old ones.

A big babel of different properties have been defined to guarantee that a system is secure. Most problems arise because it is necessary to face up to the heterogeneity of administration domains and untrustability of connections, due to geographic distribution; communications between nodes have to be guaranteed, both by making it possible to identify partners during the sessions and by preserving secrecy and integrity of the data exchanged. All the above calls for formal methods and flexible tools to catch such properties.

Specifications for message exchange, called security protocols, have been defined relying on cryptographic algorithms. Even though carefully designed, protocols may have flaws, allowing malicious agents or intruders to violate security. An intruder gaining some control over the communication network can intercept, forge and invent messages. In this way, it may convince
agents to reveal sensitive information (confidentiality problems) or to believe it is one of the legitimate agents in the session (authentication problems).

Authentication, a main issue in security, has different purposes depending on whom or which it refers to. For example, entity authentication is related to the verification of an entity’s claimed identity [16], while message authentication should make it possible for the receiver of a message to ascertain its origin [23]. All recent formalizations of these different aspects of authentication (see, e.g. [1,7,25,14,15,18,24]) are crucial for proving the corresponding properties, sometimes supported by mechanical tools (see, e.g. [1,11,12,17,19,21]).

A typical approach presented in the literature follows. First, a protocol is specified in a certain formal model; next it is shown to enjoy the desired properties, regardless of its operating environment; the protocol is then plugged into an unreliable context, possibly harbouring a hostile intruder, and proved correct; finally the protocol is implemented. Too often however, security objectives, e.g. authentication, are neglected in the design phase and are recovered later on, with obvious drawbacks. We think instead that security issues should directly influence the design of the specification languages.

We shall use here a pure calculus for concurrent and distributed systems. As it does not usually offer primitives for enforcing security, we propose a way of enhancing it with the missing mechanisms. So, we keep the advantage of reasoning on authentication and security from a formal, abstract point of view, and we can re-use existing automatic tools that support proofs of properties. Actually, we slightly extend the spi calculus [1,2], that has cryptographic primitives, and we model protocols as networks of processes, called principals (see Section 2). We give this calculus certain kinds of semantics, exploiting the built-in mechanisms for authentication, introduced in [3]. Our mechanisms (see Section 3) enable us to abstract from the various implementations/specifications of authentication, and to obtain idealized protocols which are “secure by construction”. Our protocols, rather their specifications can then be seen as a reference for proving the correctness of “real” protocols, thus bringing security issues on the stage as early as possible.

Our first mechanism is partner authentication and guarantees each principal A to engage an entire run session with the same partner B. Essentially, the semantics provides a way of “localizing” a channel for A and B, so that they accept sensitive communications on this localized channel, only. In particular, a receiver can localize who sent him a message. The key ingredient is the so-called relative address of A with respect to B [3]. Intuitively, this represents the path between A and B in (an abstract view of) the network (as defined by the syntax of the calculus). Relative addresses are not available to the users of the calculus: only the abstract machine of the calculus, defined by its semantics, is allowed to handle them. Our solution assumes that the implementation of the communication primitives has a reliable mechanism to control and manage relative addresses. In some real cases this is possible, e.g. if the network management system filters every access of a user to the network as done in a LAN or in a virtual private network. This may not be the case in many other situations. Anyway, relative addresses can be implemented by storing the actual address of processes in selected, securely encrypted parts of message headers (cf. IPsec [26]). Also our second mechanism, called message authentication [4,3], exploits relative addresses: a datum belonging to a principal A is seen by B as “localized” in the local space of A. So, our primitive enables the receiver of a message to ascertain its origin, i.e. the process that created it.

Armed with the sketched primitives (see Section 4), we give the abstract version of the protocol under consideration, which has the desired authentication properties “by construction”. A more concrete version of the protocol possibly involves encryptions, nonces, signatures and the like. It gives security guarantees, whenever its behaviour turns out to be similar, in a technical sense, to that of the abstract specification. Often, the comparison of process behaviour is done by resorting to testing equivalence [10,6]: the intuition is that two processes exhibit the same behaviour if an external process interacting with each of them cannot distinguish them. The concrete version of a protocol is secure if its behaviour cannot be taken apart from the one of the abstract version. We shall exemplify our technique in Section 5 on some simple protocols.

Our notion directly derives from a variant of Non-Interference, called NDC and applied to protocol analysis in [15,14,13]. Note also that we refine here Abadi’s and Gordon’s approach [1] of comparing cryptographic protocols with secure-by-construction specifications.
The process

The null process

cryptions

Here, terms can be names, variables and can also be defined according to the BNF-like grammars in Table 0.

with cryptographic primitives. Terms and processes are dealt with and more explanatory details can be found at the URL http://www.di.unipi.it/~chiara/publ-
dt/textFGCS.ps.

2. The spi calculus

Below, we intuitively recall a simplified version of the spi calculus [1,2], that extends the π-calculus [20] with cryptographic primitives. Terms and processes are defined according to the BNF-like grammars in Table 0. Here, terms can be names, variables and can also be encryptions \( \{M_1, \ldots, M_k\}_N \), representing the ciphertext obtained by encrypting \( M_1, \ldots, M_k \) under the key \( N \), using a shared-key cryptosystem such as DES [22].

The terms of the full calculus are also pairs, zero and successors of terms; extending our proposal to them is easy. We assume to have perfect cryptography, i.e. the only way to decrypt an encrypted message is knowing the corresponding key. Most of the processes constructs should be familiar from earlier concurrent calculi: I/O actions on \( \tau \) results.

Example 1. Define the system \( P \), with a private name \( K_{sys} \), as the parallel composition of \( A_1 \) and \( B_1 \), the replications of processes \( A_1 \) and \( B_1 \).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>-</td>
<td>Nil</td>
</tr>
<tr>
<td>( M )</td>
<td>-</td>
<td>Input</td>
</tr>
<tr>
<td>( \pi )</td>
<td>-</td>
<td>Parallel composition</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>-</td>
<td>Matching</td>
</tr>
<tr>
<td>( P )</td>
<td>-</td>
<td>Replication</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>-</td>
<td>Parallel composition</td>
</tr>
<tr>
<td>( M )</td>
<td>-</td>
<td>Matching</td>
</tr>
<tr>
<td>( P )</td>
<td>-</td>
<td>Replication</td>
</tr>
<tr>
<td>( \tau )</td>
<td>-</td>
<td>Nil</td>
</tr>
<tr>
<td>( \pi )</td>
<td>-</td>
<td>Input</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>-</td>
<td>Parallel composition</td>
</tr>
<tr>
<td>( M )</td>
<td>-</td>
<td>Matching</td>
</tr>
<tr>
<td>( P )</td>
<td>-</td>
<td>Replication</td>
</tr>
<tr>
<td>( \tau )</td>
<td>-</td>
<td>Nil</td>
</tr>
<tr>
<td>( \pi )</td>
<td>-</td>
<td>Input</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>-</td>
<td>Parallel composition</td>
</tr>
<tr>
<td>( M )</td>
<td>-</td>
<td>Matching</td>
</tr>
<tr>
<td>( P )</td>
<td>-</td>
<td>Replication</td>
</tr>
</tbody>
</table>

Table 1

The syntax of the spi-calculus

\[ L, M, N := \]

\[ \{a, b, c, k, m, n\} \]

\[ \{x, y, z, w\} \]

\[ \{M_1, \ldots, M_k\}_N \]

\[ \text{Names} \]

\[ \text{Variables} \]

\[ \text{Encryption} \]

\[ \text{Output} \]

\[ \text{Match} \]

\[ \text{Replica} \]

\[ \text{Case } L \text{ of } \{x_1, \ldots, x_k\} \text{ in } P \]

\[ \text{Decryption} \]
to the occurrences of the parallel operator in parallel compositions, and label their arcs with tag processes denote the left (respectively right) component of (respectively right) branches of a tree of sequential processes.

We can write the replication !A1 as it possesses the needed key. As a result, M replaces w in the residual of the calculus). More precisely, consider the abstract syntax trees of processes, built using the binary parallel composition operator. Given a process R, the nodes of its tree (see, e.g. Fig. 1) correspond to the occurrences of the parallel operator in R, and its leaves are the sequential components of R (roughly, those processes whose top-level operator is a prefix or a summation or a replication). Assume then the left (respectively right) branches of a tree of sequential processes denote the left (respectively right) component of parallel compositions, and label their arcs with tag ||. The set of relative addresses, ranged over by l, is

\[ A = \{ \theta_0, \theta_1 : \theta_0 = || \theta_0' \Rightarrow \theta_1 = ||_{1,0} \theta_0', i = 0, 1 \} \]

where \( \theta_0, \theta_0' \in \{ ||, \}| | \}^* \). Technically, relative addresses can be inductively built while deducing transitions, when a proved semantics is used [8,9], in which labels of transitions encode (portions of) their deduction tree [5].

In Fig. 1, the address of \( P_1 \) relative to \( P_3 \) is \( l = ||_{3,0} || ||_0 \). Notably, even if another process \( R \) possesses the channel \( c \), R cannot use it to communicate with \( P \) (or \( Q \), because its relative address with respect to \( P \) is \( l \) (not \( l^{-1} \)); additionally, \( R \) cannot alter it. Consequently, neither \( R \) nor any bos-
tile process can interfere at all with \( P \) and \( Q \) while they communicate.

Of course, processes do not always know a priori which will be the partners’ relative addresses. So, we shall also index a channel with a variable \( \lambda \) to be instantiated at due time by a relative address. A process \( P \) uses a channel \( c_\lambda \), whenever it plays, e.g. the role of sender, and wants to communicate for the first time with, say a server \( S \). Our semantic rules will take care of instantiating \( \lambda \) with the address of \( P \) relative to \( S \) during the communication. From that point on, \( P \) and \( S \) will keep communicating for the entire session, using their relative addresses. For example, let the process \( P_1 \) in Fig. 1 sends \( b \) along \( a_1 \) and become \( P'_1 \), i.e. \( P_1 = \[ a(b) \] \cdot P'_1 \), with \( l = |[a]| \cdot |[b]| \cdot |[]| \). So, \( P_1 \) wants to communicate with \( P_3 \), as its relative with respect to \( P_1 \) is indeed \( l \). Assume instead that \( P_1 \) wants to receive a value from an anonymous partner, among which \( P_0 \). Process \( P_0 \) shall then use the channel \( a_0 \) not yet localized, firing an input on it, e.g. \( a_0(x) \cdot P'_0 \). Since \( P_0 \) knows the address of its partner, while \( P_1 \) does not, our operational semantics will make sure that (i) for \( P_1 \), the output matches only an input executed by the process reachable through the relative address \( l \), (ii) for the receiver \( P_1 \), the variable \( \lambda \) is instantiated, during the communication, to the address \( l^{-1} \) of the sender \( P_0 \), with respect to \( P_1 \). From this point on, \( P_1 \) can use the channel \( a_1 |[a]| \cdot |[b]| \cdot |[]| \) to communicate with \( P_3 \), only. This mechanism resembles the connection to a new non-certified host using the ssh client, or an ss1-based web browser. The client has no guarantee of the identity of the server, so it asks the user whether she really wants to connect to the server. From that point on, however, the connection will be secured and no other entity will be able to participate in the communication.

### 3.2. Message authentication

Our second authentication primitive is for message authentication [4], and enables the receiver of a message to ascertain its origin, i.e. the process that created it. It exploits a mechanism, originally presented in [3], also based on relative addresses, that permits to define and handle local spaces of names. Before introducing our new primitive, we shall illustrate how names are linked to their creators through a simple example. For the sake of presentation, we shall use in this part no localized channels.

Let \( P_1 \) in Fig. 1 be now \( (\nu \alpha \pi n) \cdot P'_1 \). It sends its private name \( n \) to \( P_0 = \alpha(x) \cdot P'_0 \), which is enriched, while travelling on the network, with the relative address of \( P_1 \), its sender and creator, with respect to its receiver \( P_3 \). So, \( P_1 \) receives it as \( |[a]| \cdot |[x]| \cdot |[], n| = l^{-1}n \). As a matter of fact, the address \( l^{-1}n \) acts as a reference to the local space of names of \( P_3 \). Now suppose that \( P_2 \) forwards to \( P_3 \) the name \( l^{-1}n \) just received. To maintain the identity of names it suffices to keep the reference to their creators, in this case, the one to \( P_3 \). So, the address \( l^{-1}n \) will be replaced by a new relative address, that of \( P_3 \) with respect to \( P_2 \), i.e. \( |[a]| \cdot |[x]| \cdot |[], n| \). Thus, \( P_2 \) correctly refers to the name \( n \) of \( P_3 \) as \( |[a]| \cdot |[x]| \cdot |[], n| \). Both the construction of these references and their updating is done through a suitable address composition operator \( \nu \) . Its inductive definition only requires very minor extensions to the analogous operator of [3] to take care of terms.

We can now briefly recall our second authentication primitive. It is of the form \( [\theta M \nu \epsilon \nu \pi N] \) and is akin to the matching operator. This “address matching” is passed only if the relative addresses of the two localized terms under check coincide, i.e. \( l = l' \). For instance, if \( P_0 = (\nu d \pi d) \cdot P'_0 \), \( P_0 = \nu (\nu d \pi d) \) and \( P_1 = \alpha(x) \cdot \langle |[a]| \cdot |[x]| \cdot |[], |[]| \rangle \), then \( P_1 \) will be executed only if a name \( d \) coming from \( P_0 \) will replace \( x \), as the relative address of \( P_3 \) with respect to \( P_1 \) is \( |[a]| \cdot |[x]| \cdot |[], |[]| \). In fact, if \( P_1 \) communicates with \( P_3 \), then it will receive \( d \), with the address \( |[a]| \cdot |[x]| \cdot |[], |[]| \) and the matching cannot be passed, even if also \( P_1 \) is sending a value \( d \) (note that the mechanism of referencing terms to their creators makes \( \nu \)-conversions useless).

### 4. Implementing authentication

We model protocols as a network of principals, each playing a particular role (e.g. sender or receiver of a message). We observe the behaviour of a system \( P \) plugged in any environment \( E \), assuming that \( P \) and \( E \) can communicate each other on the channels they share. More precisely, \( E \) can listen on these channels, forge messages and send them on the network, possibly interfering with the behaviour of \( P \). A specification of a certain protocol, represented by \( P \), gives security guarantees, whenever its behaviour is not altered by the presence of \( E \), in a sense made clear later on.
For each protocol $P$, we present an abstract version of $P$, written using the above sketched primitives. We shall show that this version has the desired authentication properties "by construction", even in parallel with $E$. Then, we check the abstract protocol against a more concrete version, possibly involving standard cryptographic operations (e.g. encryptions, nonces, etc.). In other words, we compare the behaviour of the two versions, and we say that the concrete one is secure, whenever it exhibits the same behaviour of the abstract version. Technically, we adopt the notion of testing equivalence \cite{10,6}, where the behaviour of a protocol version is considered equivalent whenever it exhibits the same behaviour of a concrete version. In doing this, we propose to take clearly apart the graphic operations (e.g. encryptions, nonces, etc.). In other words, we compare the behaviour of the two versions, and we say that the concrete one is secure, whenever it exhibits the same behaviour of the abstract version. Technically, we adopt the notion of testing equivalence \cite{10,6}, where the behaviour of a protocol version is considered equivalent whenever it exhibits the same behaviour of a concrete version. As a matter of fact, here we push a bit further Abadi and Gordon’s \cite{1} idea of correcting a protocol if the environment cannot have any influence on its continuation. More precisely, let $P = A \parallel B$, be an abstract secure-by-construction protocol and $P = A \parallel B$ be a (bit more) concrete (cryptographic) protocol. Suppose also that both $B$ and $B_{c}$, after the execution of the protocol, continue with some activity, say $B'$. Then, we require that an external observer should not detect any difference on the behaviour of $B'$ if an intruder attacks the protocols: for all intruders $E$, we require that $A \parallel B_E$ is equivalent to $A \parallel B_{E'c}$. When this holds we say that $P$ securely implements $P_{c}$.

In doing this, we propose to take clearly apart the tester $T$ from the intruder $E$, so that we can completely abstract from the specific messages exchanged and focus only on the "effects" of the protocol execution. More precisely, we divide the protocol into two separate sequential parts: a message exchange part and a continuation part, each using different channels. So, our comparison focuses on what happens after the protocol has been executed. Technically, the comparison requires to hide the protocol message exchanges and the attacker activity. This is crucial as abstract protocols are never equivalent to their implementation if message exchanges were observed.

Basically, authentication violations are revealed by observing the address of the received message. As our testers can directly compare message addresses through our primitive of address matching, they can detect the origin of messages and check message authentication. Summing up, our authentication primitives provide secure-by-construction (abstract) protocols, so one first implements them by using, e.g. cryptography, and checks then if the two versions are testing equivalent. If successful, one has formally proved that a certain protocol $P$ implements an abstract protocol $P'$ regardless of the particular message exchange. Equally important, $P$ will always guarantee message authentication in every environment, even if harbouring hostile processes.

5. Examples

Among the applications of our approach there are authentication and freshness. Below, we take some toy protocols and check they enjoy these properties, even in presence of multi-sessions. Reasoning is straightforward in these admittedly simple examples, but we are confident that our ideas and techniques easily scale up to more complicate protocols.

Our first example has a simple single-session protocol, where $A$ sends a freshly generated message $M$ to $B$. Suppose that $B$ requires authentication of the message, i.e. that $M$ has indeed been sent by $A$. We abstractly denote this as follows, according to the standard, informal protocol narration:

Message 1 $A_{\text{sec}} \xrightarrow{M} B : M(A \text{ freshly generates } M)$

Of course, to guarantee $B$ is communicating with $A$, some trusted information has to be supplied about the partners. In real protocols this is achieved, e.g. through a password or a key known by $A$ (and/or $B$) only. We use instead the location of the entity to be authenticated, exploiting our partner authentication primitive.

Rather than giving the specification for the single-session protocol, we directly present the more interesting case, in which $m$ copies of the sender $A$ want to communicate with $m$ copies of the receiver $B$, namely its multi-session version. Note that freshness of each message $M$ in each session is enforced by the restriction operator $\nu M$. Authentication protocols require the bootstrapping of security via some assumed-secure means. Here the bootstrapping of authentication is abstracted into a startup primitive that is executed before the message-exchange and produces, in a out-of-band and trusted way, the required locations. This primitive
is the following simple macro:

\[ m_{\text{startup}}(t_A, t_B, B) \triangleq (\nu s)(\langle \pi_A(x) \cdot A!s(x) \cdot B \rangle) \]

The two processes initiate the startup by a communication over \( s \), and replicate through the "!" operator, originating many pairs of instances of \( A \) and \( B \), each playing a single, independent session. We can prove that, for all attackers \( E \), in every execution of the process \( m_{\text{startup}}(\lambda_A, A, \lambda_B, B) | E \), the two variables \( \lambda_A \) and \( \lambda_B \) only get as values the relative address of a single instance of \( B \) with respect to single instance of \( A \) and vice versa, so correctly localizing the channel \( s \). Indeed, two such variables, arising from two different sessions, never point to the same process. We now define the multi-session specification as follows:

\[ P^m = m_{\text{startup}}(\nu s, A, \lambda_A, B) \]

\[ \begin{align*}
A &= (s(M [M] M) 0 & \\
B &= axz_{A,B} \langle (z) \rangle
\end{align*} \]

(Note that the channel for \( A \) is not localized, due to the empty relative address \( s \).

After the startup phase, each copy of \( B \) waits a message \( z \) exactly from the location of one copy of \( A \) (\( B(z) \) is a process parametric w.r.t. \( z \)). This protocol is secure-by-construction. In fact, it is possible to prove that \( P^m \) enjoys the following two properties.

- **Authentication** When the continuation of an instance (of \( B \)) \( B'(\langle \delta \cdot \delta' N \rangle) \) is activated, \( \lambda_B \cdot \delta' \) must be the relative address of an instance of \( A \) with respect to the actual instance of \( B \), i.e. each copy of \( B \) receives a message enriched with the relative address of one of the copies of \( A \).

- **Freshness** For every pair of activated instances of continuations (of \( B \)) \( B'(\langle \delta \cdot \delta' N \rangle) \) and \( B'(\langle \delta \cdot \delta'' N \rangle) \) it must be \( \delta' \neq \delta'' \), i.e. the two messages have been originated by two different instances of the sender \( A \). (A more general check involves the addresses of the copies with respect to the replication process that originated them, and the address composition \( \cdot \).

Consider now a variant of the above protocol in which cryptography is used to provide authentication of the message.

**Message 1** \( A \rightarrow B : \langle M \rangle_{K_{AB}} \)

A possible specification in our calculus is the process given in Example 1, where each copy of \( A \) encrypts \( M \) to prevent anybody else to substitute a different message for \( M \). These encryptions however do not suffice: \( P^m \) is prone to replay attacks, and thus it does not implement \( P^m \). In fact, an attacker \( E = cx[\pi(x) \cdot \pi(x)] \) can intercept \( M_{K_{AB}} \) and replay it twice. Consequently, two different copies of \( B(w) \), continuations of \( R \), may be instantiated with two messages originated by the same copy of \( A \). Technically, the addresses of \( A \) relative to the two copies of \( B(w) \) are \( [1][m][s][w] \) and \( [1][m][s][w] \), respectively. This reflects that both messages have a reference to the same process where \( M_{K_{AB}} \) comes originally from, and a freshness flow is detected.

A correct implementation of the authentication protocol \( P^m \) follows. It exploits a typical challenge-response mechanism to guarantee authentication, where \( N \), the challenge, is a freshly generated one. Indeed, it is not difficult to prove that \( P_2 \) is testing equivalent to \( P^m \), and thus it is a secure implementation. Note the protocol is correct, because \( A \) and \( B \) do not play both the roles of initiator and responder in the same session; did they, a reflection attack could only be avoided by including also the principal identifier in the ciphertext.

**Message 1** \( B \rightarrow A : N \)

**Message 2** \( A \rightarrow B : [M, N]_{K_{AB}} \)

\[ \begin{align*}
P_2 &= (s[K_{AB}] A_2)[B_2] & \\
A_2 &= (s[Mx(ns)] \{ [M, ns] \}_{K_{AB}}) 0 & \\
B_2 &= (s[Nx[s]] - c(x) \cdot \text{case } \{ [z, w] \}_{K_{AB}}) & \\
in[w = X] B'(z)
\end{align*} \]

**Acknowledgement**

Work partially supported by EU-projects DEGAS (IST-2001-32072) and MyThS (IST-2001-32617).

**References**


Chiara Bodei received the Laurea degree in Computer Science in 1994 and the Ph.D. degree in Computer Science in 2000, both from the University of Pisa, Italy. Since July 2001, she is Professor Assistant of Computer Science at the University of Pisa. Her research interests are related to security, in particular to the static analysis of security properties of concurrent and mobile systems. She is also interested in the semantics of concurrency.

Pierpaolo Degano has been full professor of Computer Science since 1990 and, since 1993, has been at the Department of Computer Science, University of Pisa. From the University of Pisa, Italy. Since July 2001, she is Professor Assistant of Computer Science at the University of Pisa. Her research interests are related to security, in particular to the static analysis of security properties of concurrent and mobile systems, systems biology, semantics and concurrency, methods and tools for program verification and evaluation, and programming tools.


Riccardo Focardi received the Laurea degree in Computer Science in 1993 and the Ph.D. degree in Computer Science in 1998, both from the University of Bologna, Italy. Since December 2002, he is associate pro-

fessor at the Department of Computer Science of the University “Ca Foscari” of Venice, Italy. His research interests are related to security, in particular to the formalization and verification of system and network security properties. He is also active in the area of semantics of concurrent languages and static analysis.


Corrado Priami is full professor of computer science at the University of Trento. He obtained his laurea and Ph.D. degrees in computer science from the University of Pisa, visited as associate researcher the LIX, Ecole Politechnique in Paris (1993) and the Ecole Normale Superieure, Paris under an EC TMR grant (1996). He was researcher and associate professor at the University of Verona (1997–2001). He published over 70 papers in international journal and conferences and leads the bioinformatics group in Trento. His research covers computational methods for systems biology, languages and models for mobile and distributed systems, operational semantics, automatic verification tools, performance prediction, security, integration of qualitative and quantitative analysis. He is involved in national projects as task leader and site leader in the area of formal methods and semantics, and he participated to previous ESPRIT projects. He is currently coordinator of the EU IST-2001-32072 project DEGAS. He was Chair of international events and participated to program and organizing committees.