Authentication Primitives for Protocol Specifications *

Chiara Bodei[1] Pierpaolo Degano[1] Riccardo Focardi[2] Corrado Priami[3]

[1] Dipartimento di Informatica, Università di Pisa
Via Filippo Buonarroti, 2, I-56127 Pisa, Italy – {chiara,degano}@di.unipi.it
[2] Dipartimento di Informatica, Università Ca' Foscari di Venezia,
Via Torino 155, I-30173 Venezia, Italy –
focardi@dsi.unive.it
[3] Dipartimento di Informatica e Telecomunicazioni, Università di Trento
Via Sommarive, 1438050 Povo (TN), Italy – priami@science.unitn.it

Abstract. We advocate here the use of two authentication primitives we recently propose in a calculus for distributed systems, as a further instrument for programmers interested in authentication. These primitives offer a way of abstracting from various specifications of authentication and obtaining idealized protocols "secure by construction". We can consequently prove that a cryptographic protocol is the correct implementation of the corresponding abstract protocol; when the proof fails, reasoning on the abstract specification may drive to the correct implementation.

1 Introduction

Security in the times of Internet is something people cannot do without. Security has to do with confidentiality, integrity and availability, but also with non-repudiation, authenticity and even more, depending on the application one has in mind. The technology of distributed and parallel systems and networks as well influences security, introducing new problems and scenarios, and updating some of the old ones.

A big babel of different properties and measures have been defined to guarantee that a system is secure. All the above calls for formal methods and flexible tools to catch the elusive nature of security.

Mostly, 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 the secrecy and integrity of the data exchanged. To this end specifications for message exchange, called security protocols, are defined on the basis of 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 is able to intercept or forge or invent messages. In this way the intruder may convince agents to reveal sensitive information (confidentiality's problems) or to believe it is one of the legitimate agents in the session (authentication's problems).

* Work partially supported by EU-project DEGAS (IST-2001-32072) and by Progetto MIUR Metodi Formali per la Sicurezza (MEFISTO).

Authentication is one of the main issues in security and it can have different purposes depending on the specific application considered. For example, *entity authentication* is related to the verification of an entity's claimed identity [20], while *message authentication* should make it possible for the receiver of a message to ascertain its origin [28]. In recent years there have been some formalizations of these different aspects of authentication (see, e.g., [1, 8, 14, 16, 17, 21, 27]). These formalizations are crucial for proofs of authentication properties, that sometimes have been automatized (see, e.g. [11, 18, 23, 22, 25]).

A typical approach presented in the literature is the following. First, a protocol is specified in a certain formal model. Then the protocol is shown to enjoy the desired properties, regardless of its operating environment, that can be unreliable, and can even harbour a hostile intruder.

We use here basic calculi for modelling concurrent and mobile agents. In particular, we model protocols as systems of processes, called *principals* or *parties*. Using a pure calculus allows us to reason on authentication and security from an abstract point of view. Too often, security objectives, like authentication, are not considered in the very design phase and are instead approximately recovered after it. The ideal line underlying our approach relies on the conviction that security should directly influence the design of programming languages, because languages for concurrent and distributed systems do not naturally embed security.

In particular, we here slightly extend the spi calculus [1, 2], a language for modelling concurrent and distributed agents, endowed with cryptographic primitives. We give this calculus certain kinds of semantics, exploiting the built-in mechanisms for authentication, introduced in [4]. Our mechanisms enable us to abstract from the various implementations/specifications of authentication, and to obtain idealized protocols which are "secure by construction". Our protocols, or rather their specifications can then be seen as a reference for proving the correctness of "real" protocols.

In particular, our first mechanism, called *partner authentication* [4], 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 to A and B, so that the partners accept sensitive communications on this localized channel, only. In particular, a receiver can localize the principal that sent him a message. Such a localization relies on the so-called *relative address* of A with respect to B. 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: they are used by the abstract machine of the calculus only, defined by its semantics.

Our solutions assume 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 it happens in a LAN or in a virtual private network. This may not be the case in many other situations. However, relative addresses can be built by storing the actual address of processes in selected, secure parts of message headers (cf. IPsec [19]).

Also our second mechanism, called *message authentication* [6, 4], 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.

The above sketched primitives help us to 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 to that of the abstract specification. A classical process algebraic technique to compare the behaviour of processes is using some notion of equivalence: the intuition is that two processes have the same behaviour if no distinction can be detected by an external process interacting with each of them. The concrete version of a protocol is secure if its behaviour cannot be distinguished from the one of the abstract version. This approach leads to testing equivalence [10, 7] and we shall follow it hereafter.

Our notion directly derives from the Non-Interference notion called NDC that has been applied to protocol analysis in [17, 16, 15]. Note also that the idea of comparing cryptographic protocol with secure-by-construction specifications is also similar to the one proposed in [1] where a protocol is compared with "its own" secure specification. We are indeed refining Abadi's and Gordon's approach [1]: the secure abstract protocol here is unique (as we will show in the following) and based on abstract authentication primitives. On the contrary, in [1] for each protocol one needs to derive a secure specification (still based on cryptography) and to use it as a reference for proving authentication.

The paper is organized as follows. The next section briefly surveys our version of the spi calculus. Section 3 intuitively presents our authentication primitives, Section 4 introduces our notion of correct implementation. Finally, Section 5 gives some applications.

2 The Spi Calculus

Syntax. In this section we intuitively recall a simplified version of the spi calculus [1, 2]. In the full calculus, terms can also be pairs, zero and successors of terms. Extending our proposal to the full calculus is easy. Our version of the calculus extends the π-calculus [24], with cryptographic primitives. Here, terms can be names, variables and can also be structured as pairs (M_1, M_2) or encryptions $\{M_1, \ldots, M_k\}_N$. An encryption $\{M_1, \ldots, M_k\}_N$ represents the ciphertext obtained by encrypting $M_1, \ldots, M_k$ under the key N, using a shared-key cryptosystem such as DES [9]. 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 constructs, parallel composition, restriction, matching, replication. We give below the syntax and, afterwards, we intuitively present the dynamics of processes. Terms and processes are defined according to the following BNF-like grammars.

$$\begin{array}{ll}
L, M, N ::= & \mathbf{terms} \\
\quad a, b, c, k, m, n & names \\
\quad x, y, z, w & variables \\
\quad \{M_1, \ldots, M_k\}_N & shared\ encryption
\end{array}$$

$P, Q, R ::=$	**processes**
$\mathbf{0}$	*nil*
$\overline{M}\langle N\rangle.P$	*output*
$M(x).P$	*input*
$(\nu m)P$	*restriction*
$P\|P$	*parallel composition*
$[M = N]P$	*matching*
$!P$	*replication*
$case\ L\ of\ \{x_1, \ldots, x_k\}_k\ in\ P$	*shared−key decryption*

- The null process $\mathbf{0}$ does nothing.
- The process $\overline{M}\langle N\rangle.P$ sends the term N on the channel denoted by M (a name, or a variable to be bound to), provided that there is another process waiting to receive on the same channel. Then behaves like P.
- The process $M(x).P$ is ready to receive an input N on the channel denoted by M and to behave like $P\{N/x\}$, where the term N is bound to the variable x.
- The operator $(\nu m)P$ acts as a static declaration (i.e. a binder for) the name m in the process P that it prefixes. The agent $(\nu m)P$ behaves as P except that I/O actions on m are prohibited.
- The operator $|$ describes parallel composition of processes. The components of $P|Q$ may act independently; also, an output action of P (resp. Q) at any output port $\overline{M}$ may synchronize with an input action of Q (resp. P) at M. In this case, a silent action τ results.
- Matching $[M = N]P$ is an `if-then` operator: process P is activated only if $M = N$.
- The process $!P$ behaves as infinitely many copies of P running in parallel, i.e. it behaves like $P \mid !P$.
- The process $case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P$ attempts to decrypt L with the key N. If L has the form $\{M_1, \ldots, M_k\}_N$, then the process behaves as the process P, where each x_i has been replaced by M_i, i.e.as the process $P\{M_1/x_1, \ldots, M_k/x_k\}$. Otherwise the process is stuck.

The operational semantics of the calculus is a labelled transition system, defined in the SOS, logical style. The transitions are represented as $P \xrightarrow{\tau} P'$, where the label corresponds to a silent or internal action action τ that leads the process P in the process P'. To give the flavour of the semantics, we illustrate the dynamic evolution of a simple process S. For more details, see [4].

Example 1. In this example, the system S is given by the parallel composition of the replication (of process P) $!P$ and of the process Q.

$$\begin{aligned} S &= !P \mid Q \\ P &= \overline{a}\langle\{M\}_k\rangle.\mathbf{0} \\ Q &= a(x).case\ x\ of\ \{y\}_k\ in\ Q' \\ Q' &= (\nu h)(\overline{b}\langle\{y\}_h\rangle.\mathbf{0} \mid R) \end{aligned}$$

$!P$ represents a source of infinitely many outputs on a of the message M encrypted under k. Therefore it can be rewritten as $P \mid !P = \overline{a}\langle\{M\}_k\rangle.\mathbf{0} \mid !P$. So, we have the following part of computation:

$$S \xrightarrow{\tau} \mathbf{0} \mid !P \mid case\ \{M\}_k\ of\ \{y\}_k\ in\ Q' \xrightarrow{\tau} \mathbf{0} \mid !P \mid (\nu h)(\overline{b}\langle\{M\}_h\rangle.\mathbf{0} \mid R)$$

In the first transition, Q receives on channel a the message $\{M\}_k$ sent by P and $\{M\}_k$ replaces x in the residual of Q. In the second transition, $\{M\}_k$ can be successfully decrypted by the residual of Q, with the correct key k and M replaces y in Q'. The effect is to encrypt M with the key h, private to Q'. The resulting output $\overline{b}\langle\{M\}_h\rangle$ can occur to be matched by some input in R.

3 Authentication Primitives

Before presenting our authentication mechanisms [4], it is convenient to briefly recall the central notion of *relative address* of a process P with respect to another process Q within a network of processes, described in our calculus. A relative address represents the path between P and Q in (an abstract view of) the network (as defined by the syntax of the calculus). More precisely, consider the abstract syntax trees of processes, built using the binary parallel composition as the main 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). Assuming that the left (resp.

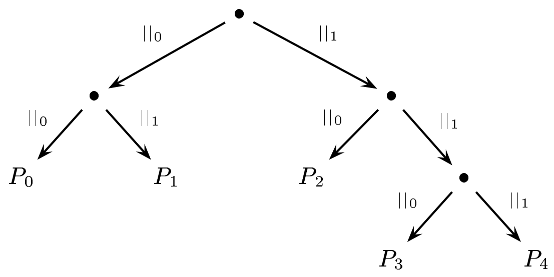

Fig. 1. The tree of (sequential) processes of $(P_0|P_1)|(P_2|(P_3|P_4))$.

right) branches of a tree of sequential processes denote the left (resp. right) component of parallel compositions, then label their arcs with tag $||_0$ (resp. $||_1$).

Tecnically, relative addresses can be inductively built while deducing transitions, when a proved semantics is used [13, 12], in which labels of transitions encode (a portion of) their deduction tree.

We recall the formal definition of relative addresses [5].

Definition 1 (relative addresses).
Let $\vartheta_i, \vartheta'_i \in \{||_0, ||_1\}^*$, *let* ϵ *be the empty string. Then, the set of* relative addresses*, ranged over by* l*, is*

$$\mathcal{A} = \{\vartheta_0 \bullet \vartheta_1 : \vartheta_0 = ||_i \vartheta'_0 \Rightarrow \vartheta_1 = ||_{1-i} \vartheta'_1, i = 0, 1\}.$$

For instance, in Fig. 1, the address of P_3 relative to P_1 is $l = ||_0||_1 \bullet ||_1||_1||_0$ (read the path upwards from P_1 to the minimal common predecessor and reverse, then downwards to P_3). So to speak, the relative address points back from P_1 to P_3.

Note that the relative address of P_3 with respect to P_1 is $||_1||_1||_0 \bullet ||_0||_1$ that we write also as l^{-1}. When two relative addresses l, l' both refer to the same path, exchanging its source and target, we call them compatible. Formally, we have the following definition.

Definition 2. *A relative address* $l' = \vartheta' \bullet \vartheta$ *is* compatible *with* l*, written* $l' = l^{-1}$*, if and only if* $l = \vartheta \bullet \vartheta'$.

We are now ready to introduce our primitives that induce a few modifications to the calculus surveyed above. Note that we separately present below the two primitives, but they can be easily combined, in order to enforce both kinds of authentication.

3.1 Partner Authentication

We can now intuitively present our first semantic mechanism, originally presented in [4]. Essentially, we bind sensitive inputs and outputs to a relative address, i.e. a process P can accept communications on a certain channel, say c, only if the relative address of its partner is equal to an *a priori* fixed address l. More precisely, channels may have a relative address as index, and assume the form c_l. Now, our semantics will ensure that P communicates with Q on c_l if and only if the relative address of P with respect to Q is indeed l (and that of Q with respect to P is l^{-1}). Notably, even if another process $R \neq Q$ possesses the channel c_l, R cannot use it to communicate with P, because relative addresses are not available to the users. Consequently, the hostile process R can never interfere with P and Q while they communicate, as the relative address of R with respect to Q (and to P) is not l (or l^{-1}).

Processes do not always know *a priori* which are the partners' relative addresses. So, we shall also index a channel with a variable λ, to be instantiated by a relative address, only. Whenever a process P, playing for instance the role of sender, has to communicate for the first time with another process S in the role, e.g. of server, it uses a channel c_λ. Our semantic rules will take care of instantiating λ 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.

Suppose, for instance, that in Fig. 1 the process P_3 sends b along a_l and becomes P'_3, i.e. is $\overline{a_l}\langle b\rangle.P'_3$ and that P_1 reads on a not yet localized channel a_λ a value, e.g. $a_\lambda(x).P'_1$; recall also that the relative address of P_3 with respect to the process P_1 is $l = ||_1||_1||_0 \bullet ||_0||_1$. Here P_3 knows the partner address, while P_1 does not. More precisely, for P_3 the output can only match an input executed by the process reachable from P_3 through the relative address l, while the variable λ will be instantiated, during the communication, to the address l^{-1} of the sender P_3, with respect to the receiver P_1.

From this point on and for the rest of the protocol, P_1 can use the channel $a_{||_0||_1\bullet||_1||_1||_0}$ (and others that may have the form c_λ) to communicate with P_3, only.

3.2 Message Authentication

Our second mechanism, called *message authentication*, originally presented in [6, 4], enables the receiver of a message to ascertain its origin, i.e. the process that created it. Again it is based on relative addresses.

We illustrate this further extension originally modelled in [6] through a simple example. Suppose that P_3 in Fig. 1 is now $(\nu n)\overline{a}\langle n\rangle.P_3'$. It sends its private name n to $P_1 = a(x).P_1'$. The process P_1 receives it as $||_1||_0\bullet||_1||_1||_0 n = l^{-1}n$. In fact, the name n is enriched with the relative address of P_3, its sender and creator, with respect to its receiver P_1 and the address l^{-1} acts as a reference to P_3. Now suppose that P_1' forwards to P_2 the name just received, i.e. $l^{-1}n$. We wish to maintain the identity of names, i.e., in this case, the reference to P_3. So, the address l^{-1} will be substituted by a new relative address, that of P_3 with respect to P_2, i.e. $||_1||_0\bullet||_0$. Thus, the name n of P_3 is correctly referred to as $||_1||_0\bullet||_0 n$ in P_2. This updating of relative addresses is done through a suitable address composition operation (see [4] for its definition).

We can now briefly recall our second authentication primitive, $[lM \overset{@}{=} l'N]$, 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 $P_3 = (\nu d)\overline{a}\langle d\rangle.P_3'$, $P_0 = (\nu b)\overline{a}\langle b\rangle$ and $P_1 = a(x).[x = ||_0||_1\bullet||_1||_1||_0 d]P_1'$, then P_1' will be executed only if x will be replaced with a name coming from P_3, such as $||_0||_1\bullet||_1||_1||_0 n$. In fact, if P_1 communicates with P_0, then it will receive b, with the address $||_0||_0\bullet||_1||_1||_0$ and the matching cannot be passed.

4 Implementing Authentication

We model protocols as systems 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 and can send on these channels, possibly interfering with the behaviour of P.

A (specification of a certain) protocol, represented by P, gives security guarantees, whenever its behaviour it is not compromised 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 will show that this version has the desired authentication properties "by construction", even in parallel with E. Then, we check the abstract protocol against a different, more *concrete* version, possibly involving standard cryptographic operations (e.g. encryptions, nonces). In other words, we compare their behaviour. The concrete version is secure, whenever it presents the same behaviour of the abstract version. We adopt here the notion of testing equivalence [10, 7], where the behaviour of processes is observed by an external process, called tester. Testers are able to observe all the actions of systems, apart from the internal ones.

As a matter of fact, here we push a bit further Abadi and Gordon's [1] idea of considering correct a protocol if the environment cannot have any influence on its continuation. More precisely, let $P_s = A_s|B_s$ be an abstract secure-by-construction protocol and $P = A|B$ be a (bit more) concrete (cryptographic) protocol. Suppose also that both B and B_s, 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 E attacks the protocols. In other words, for all intruders E, we require that $A|B|E$ is equivalent to $A_s|B_s|E$. When this holds we say that P *securely implements* P_s. In doing this, we propose to clearly separate the observer, or tester T, from the intruder E. In particular, we let the tester T interact with the continuation B', only. Conversely, we assume that the intruder attacks the protocol, only, and we do not consider how the intruder exploits the attacks for interfering on what happens later on. This allows us to completely abstract from the specific message exchange (i.e., from the communication) and focus only on the "effects" of the protocol execution.

This allows us to compare protocols which may heavily differ in the messages exchanged. In fact, as our authentication primitives provide secure-by-construction (abstract) protocols, the idea is to try to implement them by using, e.g., cryptography. We therefore adopt testing equivalence to formally prove that a certain protocol P implements an abstract protocol P' regardless of the particular message exchange.

We can keep message exchange apart from the rest of the protocol. In our model, protocol specifications are then seen as composed of two sequential parts: a message exchange part and a continuation part, kept separate by using different channels. As said above, the comparison we use focuses on the effects of the protocol execution on the continuation, i.e., on what happens after the protocol has been executed. In other words, the comparison is performed by making invisible 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. Moreover, since authentication violations are easily revealed by observing the address of the received message, we can exploit our operator of address matching to this aim. In particular, in our notion of testing equivalence, testers have the ability of directly comparing message addresses (through address matching), thus detecting the origin of messages.

Our notion is such that if P' is a correct-by-construction protocol, specified through our authentication primitives, and P securely implements P', then also the behaviour of P in every hostile environment, i.e. plugged in parallel with any other process, will be correct.

4.1 A Notion of Secure Implementation

We give here the formal definition of testing equivalence [4] directly on the spi calculus.

We write $P \xrightarrow{\overline{m}}$ ($P \xrightarrow{m}$, resp.), whenever the process P performs an output (an input, resp.) on the channel m. When the kind of action is immaterial, we shall write $P \xrightarrow{\beta}$ and call β a *barb*.

A *test* is a pair (T, β), where T is a closed process called tester and β is a *barb*. Then, a process P *exhibits* β (denoted by $P \downarrow \beta$) if and only if we have $P \xrightarrow{\beta}$, i.e. if P

[4] Technically it is a *may*-testing equivalence.

can do a transition on β. Moreover P *converges on* β (denoted by $P \Downarrow \beta$) if and only if $P \xrightarrow{\tau}^{*} P'$ and $P' \downarrow \beta$. Now, we say that a process P *immediately passes a test* (T, β) if and only if $(P \mid T) \downarrow \beta$. We also say that a process P *passes a test* (T, β) if and only if $(P \mid T) \Downarrow \beta$.

Our testers are processes that can directly refer to addresses in the address matching operator. As an example, a tester may be the following process $T = observe(z).[z \stackrel{@}{=} ||_1||_0\bullet||_1]\beta(x)$. A tester has therefore a global view of the network, because it has full knowledge of addresses, i.e., of the locations of processes. More importantly, this feature of the testers gives them the ability to directly observe authentication attacks. Indeed a tester may check if a certain message has been originated by the expected location. As an example, T receives a message on channel $observe$ and checks if it has been originated at $||_1||_0\bullet||_1$. Only in this case, the test (T, β) is passed, as the global process (T composed with the protocol) exhibits the barb β. We call $\mathcal{T}$ the set of tester processes.

Now we define the testing preorder $\leq$: a process P is in this relation with a process Q, when each time P passes a test (T, β), then Q passes the test as well.

Definition 3. $P \leq Q$ *iff* $\forall T \in \mathcal{T}, \forall \beta : (P \mid T) \Downarrow \beta$ implies $(Q \mid T) \Downarrow \beta$.

As seen above, in our model, protocol specifications are composed of two parts: a message exchange part and a continuation part. Moreover, we assume that the attacker knows the channels that convey messages during the protocol. These channels are *not* used in continuations and can be extracted from the specification of the protocol itself. Note that the continuations may often use channels, that can also be transmitted during their execution, but never used to transmit messages.

We can now give our notion of implementation, where $C = \{c_1, \ldots, c_n\}$ is the set of all the channels used by the protocols P and P'.

Definition 4. *Let P and P' be two protocols that communicate over C. We say that P securely implements P' if and only if*

$$\forall X \in \mathcal{E}_C : (\nu c_1), \ldots, (\nu c_n)(P \mid X) \leq (\nu c_1), \ldots, (\nu c_n)(P' \mid X)$$

where $\mathcal{E}_C$ is the set of processes that can only communicate over channels in C.

Note that the names of channels in C are restricted. Moreover, we require that X may only communicate through them. These assumptions represent some mild and reasonable constraints that are useful for the application of the testing equivalence.

These assumptions have both the effect of isolating all the attacker's activity inside the scope of the restriction $(\nu c_1), \ldots, (\nu c_n)$ and of making all the message exchanges that may be performed by P and P' not observable. As a consequence, we only observe what is done after the protocol execution: the only possible barbs come from the continuations. As we have already remarked, observing the communications part would distinguish protocols based on different message exchanges even if they provide the same security guarantees. Instead, we want to verify whether P implements P', regardless of the particular underlying message exchange and of the possible hostile execution environment.

The definition above requires that when P and P' are executed in a hostile environment X, all the behaviour of P are also a possible behaviour for P'. So if P' is a correct-by-construction protocol, specified through some authentication primitives, and P securely implements P', then also P is correct, being also a behaviour for the correct-by-construction protocol P'.

As anticipated in the Introduction, this definition directly derives from the NDC notion. In particular it borrows from NDC the crucial idea of not observing both the communication and the attacker's activity.

5 Some Applications

We show how our approach can be applied to study authentication and freshness. To exemplify our proposal we consider some toy protocols. Nevertheless, we feel that the ideas and the techniques presented could easily scale up to more complicate protocols.

5.1 A Single Session

Consider a simple single-session protocol where A sends a freshly generated message M to B and suppose that B requires authentication of the message, i.e., that M is indeed sent by A. We abstractly denote this as follows, according to the standard and informal protocol narration:

$$\begin{array}{ll} (A \text{ freshly generates } M) & \\ \text{Message 1} & A \stackrel{auth}{\to} B :\ M \end{array}$$

Note that, if B wants to be guaranteed that he is communicating with A, he needs as a reference some trusted information regarding A. In real protocols this is achieved, e.g., through a password or a key known by A only. We use instead the location of the entity that we want to authenticate. In order to do this, we specify this abstract protocol by exploiting our partner authentication primitive. The generation of a fresh message is simply modelled through the restriction operator νM of our calculus.

In order to allow the protocol parties to securely obtain the location of the entity to authenticate, we define a startup primitive that exchanges the respective locations in a trusted way. This primitive is indeed just a macro, defined as follows:

$$startup(t_A, A, t_B, B) \stackrel{\Delta}{=} (\nu s)(\ \overline{s}_{t_A}\langle s\rangle.A \mid s_{t_B}(x).B\ )$$

where x does not occur in B and s does not occur in A and B. The restriction on s syntactically guarantees that communications on that channel cannot be altered by anyone else, except for A and B. This holds also when the process is executed in parallel with any possibly hostile environment E. Now, in process $startup(\lambda_A, A, \lambda_B, B)$, after the communication over the fresh channel s, variables λ_A and λ_B are securely bound to the addresses of B and A, respectively. More precisely, for each channel c_{λ_A} in A, λ_A is instantiated to the address of B w.r.t. A, while for each channel c_{λ_B} in B, λ_B is instantiated to the address of A w.r.t. B. So, on these channels, A and B can only communicate each other.

In particular, the following holds:

Proposition 1. *Consider the process $startup(\lambda_A, A, \lambda_B, B)$. Then, for all possible processes E, in any possible execution of $startup(\lambda_A, A, \lambda_B, B) \mid E$, the location variable λ_A (λ_B, resp.) can be only assigned to the relative address $||_0\bullet||_1$, of B with respect to A (to the relative address $||_1\bullet||_0$ of A with respect to B, resp.).*

Proof. *By case analysis.*

Now, we show an abstract specification of the simple protocol presented above:

$$\begin{aligned}
P &= startup(\epsilon\bullet\epsilon, A, \lambda_B, B)\\
A &= (\nu M)\overline{c}\langle M\rangle\\
B &= c_{\lambda_B}(z).B'(z)
\end{aligned}$$

Technically, using $\epsilon\bullet\epsilon$ in the place of t_A corresponds to having no localization for the channel with index t_A, e.g. $c_{\epsilon\bullet\epsilon} = c$.

After the startup phase, B waits for a message z from the location of A: any other message coming from a different location cannot be received. In this way we model authentication.

Note that locating the output of M in A (as in $A' = (\nu M)\overline{c_{||_0\bullet||_1}}\langle M\rangle$) would give a secrecy guarantee on the message, because the process A would be sure that B is the only possible receiver of M.

Due to the partner authentication, this protocol is secure-by-construction. To see why, consider its execution in a possibly hostile environment, i.e. consider $P \mid E$. By Proposition 1, we directly obtain that λ_B is always assigned to the relative address of A w.r.t. B, i.e., $||_1\bullet||_0$. Thus, the sematic rules ensure that B can only receive a value z sent by A, on the located channel $c_{||_1\bullet||_0}$. Since A only sends one freshly generated message, we conclude that z will always contain a located name with address $||_1\bullet||_0$. This means that B always receives a message which is authentic from A.

As intuitively described in Section 3, the location of the channel c in process B guarantees a form of *entity authentication*: by construction, B communicates with the correct party A. Then, since A is following the protocol (i.e., is not cheating), we also obtain a form of *message authentication* on the received message, i.e., B is ensured that the received message has been originated by A.

To further clarify this we show the two possible execution sequences of the protocol:

$$\begin{aligned}
P \mid E &= startup(\epsilon\bullet\epsilon, A, \lambda_B, B) \mid E\\
&= (\nu s)(\ \overline{s}\langle s\rangle.A \mid s_{\lambda_B}(x).B\ ) \mid E\\
&\xrightarrow{\tau} (\nu s)(\ (\nu M)\overline{c}\langle M\rangle \mid c_{||_1\bullet||_0}(z).B'(z)\ ) \mid E
\end{aligned}$$

There are now two possible moves. E may intercept the message sent by A (and then continue as E'):

$$(\nu s)(\ (\nu M)\overline{c}\langle M\rangle \mid c_{||_1\bullet||_0}(z).B'(z)\ ) \mid E \xrightarrow{\tau}$$

$$(\nu_\bullet||_0||_0 M)(\ (\nu s)(\ \mathbf{0} \mid c_{||_1\bullet||_0}(z).B'(z)\ ) \mid E')$$

The way addresses are handled makes M to be received by E' as $||_1\bullet||_0||_0 M$, that is with address of A w.r.t. E. For the same reason, the restriction on M in the target of the transition become $(\nu_{\bullet}||_0||_0 M)$.

The other possible interaction is the one between A and B:

$$(\nu s)(\ (\nu M)\overline{c}\langle M\rangle \mid c_{||_1\bullet||_0}(z).B'(z)\ ) \mid E \xrightarrow{\tau}$$

$$(\nu s)(\nu_{\bullet}||_0 M)(\ \mathbf{0} \mid B'(||_1\bullet||_0 M)\ ) \mid E$$

It is important to observe that there is no possibility for E to make B accept a faked message, as B will never accept a communication from a location which is different from $||_1\bullet||_0$.

We now show how the abstract protocol above can be used as a reference for more concrete ones, by exploiting the notion of protocol implementation introduced in the previous section.

First, consider a clearly insecure protocol in which A sends M as plaintext to B, without any localized channel.

$$\begin{aligned} P_1 &= A_1 \mid B_1 \\ A_1 &= (\nu M)\overline{c}\langle M\rangle \\ B_1 &= c(z).B'(z) \end{aligned}$$

We can prove that P_1 does not implement P, by using testing equivalence. Consider a continuation that exhibits the received value z. So, let $B'(z) = \overline{observe}\langle z\rangle$, and consider the processes $(\nu c)(P \mid E)$ and $(\nu c)(P_1 \mid E)$, where $E = (\nu M_E)\overline{c}\langle M_E\rangle$ is an attacker which sends a fresh message to B, pretending to be A. Let the tester T be the process $observe(z).[z \overset{@}{=} ||_1||_0\bullet||_1]\beta(y)$ which detects if z has been originated by E. Note that the only possible barb of the two processes we are considering is the output channel $observe$. It is clear that $(\nu c)(P_1 \mid E)$ may pass the test (T, β) while $(\nu c)(P \mid E)$ cannot pass it, thus $(\nu c)(P \mid E) \not\lesssim (\nu c)(P_1 \mid E)$. In fact, P_1 can receive the value M_E on z with the address of B_1 w.r.t. E, that it is different from the expected $||_1||_0\bullet||_1$. This counter-example corresponds to the following attack:

Message 1 $E(A) \rightarrow B:\ M_E$ E pretending to be A

We show now that the following protocol, that uses cryptography is able to provide authentication of the exchanged message (in a single protocol session):

Message 1 $A \rightarrow B:\ \{M\}_{K_{AB}}$

K_{AB} is an encryption key shared between A and B. We specify this protocol as follows:

$$\begin{aligned} P_2 &= (\nu K_{AB})(A_2 \mid B_2) \\ A_2 &= (\nu M)\overline{c}\langle\{M\}_{K_{AB}}\rangle \\ B_2 &= c(z).\mathit{case}\ z\ \mathit{of}\ \{w\}_{K_{AB}}\ \mathit{in}\ B'(w) \end{aligned}$$

Here, A_2 encrypts M to protect it. Indeed, the goal is to prevent other principals from substituting for M a different message, as it may happen in P_1. This is a correct way of implementing our abstract authentication primitive in a single protocol session. In order to prove that P_2 securely implements P, one has to show that every computation of $(\nu c)(P_2|X)$ is simulated by $(\nu c)(P|X)$, for all $X \in \mathcal{E}_c$. This is indeed the case and P_2 gives entity authentication guarantees: B_2 can be sure that it is A the sender of the message. On the other hand, we can also have a form of message authentication, as far as the delivered message w is concerned, since our testers are able to observe the originator of a message through the address matching operator.

Proposition 2. *P_2 securely implements P*

Proof. We give a sketch of the proof. We have to show that every computation of $(\nu c)(P_2|X)$ is simulated by $(\nu c)(P|X)$, for all $X \in \mathcal{E}_c$. To this purpose, we define a relation $\mathcal{S}$ which can be proved to be a barbed weak simulation. Barbed bisimulation [26] provides very efficient proof techniques for verifying may-testing preorder, and is defined as follows. A relation $\mathcal{S}$ is a barbed weak simulation if for $(P, Q) \in \mathcal{S}$:

- $P \downarrow \beta$ implies that $Q \Downarrow \beta$,
- if $P \xrightarrow{\tau} P'$ then there exists Q' s.t. $Q(\xrightarrow{\tau})^* Q'$ and $(P', Q') \in \mathcal{S}$.

The union of all barbed simulation is represented by $\trianglelefteq$. Moreover, we say that a relation $\mathcal{S}$ is a barbed weak pre-order (denoted with $\lessapprox$) if for $(P, Q) \in \mathcal{S}$ and for all $R \in \mathcal{T}$ we have $P \mid R \trianglelefteq Q \mid R$. It is easy to prove that $\lessapprox \subseteq \leq_{may}$.

We now define a relation $\mathcal{S}$ as follows:

$$(\nu c)(\nu||_0 K_{AB})(\nu||_0||_0 M)\ (\ (\tilde{A} \mid B_2) \mid X\ )\quad \mathcal{S}\quad (\nu c)(P \mid X)$$

where either $\tilde{A} = A_2$ or $\tilde{A} = \mathbf{0}$. Moreover the key K_{AB} may appear in X only in a term $||_0||_0\bullet||_1\{M\}_{K_{AB}}$, possibly as a subterm of some other composed term.

The most interesting moves are the following:

- $\tilde{A} = A_2$, and

$$\begin{array}{l}(\nu c)(\nu||_0 K_{AB})(\nu||_0||_0 M)\ (\ (A_2 \mid B_2) \mid X\ )\\ \xrightarrow{\tau} (\nu c)(\nu||_0 K_{AB})(\nu||_0||_0 M)\ (\ (\mathbf{0} \mid B'(||_0\bullet||_1 M)) \mid X\ ) = F\end{array}$$

  This is simulated as
  $(\nu c)(P \mid Y) \xrightarrow{\tau}\xrightarrow{\tau} (\nu c)(\nu||_0 M)(\ (\ \mathbf{0} \mid B'(||_0\bullet||_1 M)) \mid X\ ) = G$. It is easy to see that $F \equiv G$, since K_{AB} is not free in $B'(w)$.
- $\tilde{A} = A_2$, and

$$\begin{array}{l}(\nu c)(\nu||_0 K_{AB})(\nu||_0||_0 M)\ (\ (\ A_2 \mid B_2) \mid X\ )\\ \xrightarrow{\tau} (\nu c)(\nu||_0 K_{AB})(\nu||_0||_0 M)(\ (\ \mathbf{0} \mid B_2\ ) \mid X'\ )\end{array}$$

  Here X' intercepts the message which is exactly $||_0||_0\bullet||_1\{M\}_{K_{AB}}$. This is simulated by just idling. We indeed obtain that
  $(\nu c)(\nu||_0 K_{AB})(\nu||_0||_0 M)(\ (\ \mathbf{0} \mid B_2\ ) \mid X'\ )\ \mathcal{S}\ (\nu c)(P \mid X)$.

– $\tilde{A} = \mathbf{0}$ and

$$\begin{array}{l} (\nu c)(\nu||_0 K_{AB})(\nu||_0||_0 M)\ (\ (\ \mathbf{0} \mid B_2\ ) \mid X\ ) \\ \xrightarrow{\tau} (\nu c)(\nu||_0 K_{AB})(\nu||_0||_0 M) \\ \quad (\ (\ \mathbf{0} \mid \mathit{case}\ \theta_\bullet\theta'\{N\}_{K_{AB}}\ \mathit{of}\ \{w\}_{K_{AB}}\ \mathit{in}\ B'(w)\ ) \mid X'\ ) = F' \end{array}$$

By the hypothesis on X it must be $N = M$ and $\theta_\bullet\theta' = ||_0||_0\bullet||_1$. Thus

$$F' \equiv (\nu c)(\nu||_0 K_{AB})(\nu||_0||_0 M)\ (\ (\ \mathbf{0} \mid B'(||_0\bullet||_1 M)\ ) \mid X'\ )$$

This is simulated as for the first case above.

Since $(\nu c)(P_2|X)\ \mathcal{S}(\nu c)(P|Y)$, we obtain the thesis.

5.2 Multiple Sessions

The version of the protocol P_2 is secure if we consider just one single session, but it is no longer such, when considering more than one session. We will see this, and we will also see how to repair the above specification in order to obtain the same guarantees. Our first step is extending the $startup$ macro to the multisession case:

$$m\_startup(t_A, A, t_B, B) \stackrel{\Delta}{=} (\nu s)(\ !\overline{s}_{t_A}\langle s\rangle.A \mid !s_{t_B}(x).B\ )$$

Two processes that initiate the startup by a communication over s are replicated through the "!" operator; so there are many pairs of instances of the sub-processes A and B communicating each other. Each pair plays a single session.

The following result extends Proposition 1 to the multisession case (note that here any replication originates a new instance of the two location variables). Intuitively, the proposition below states that, when many sessions are considered, our startup mechanism is able to establish different independent runs between instances of P and Q, where no messages of one run may be received in a different run. This is a crucial point that provides freshness, thus avoiding replay of messages from a different run.

Proposition 3. *Consider the process $startup(\lambda_A, A, \lambda_B, B)$. Then, for all possible processes E, in any possible execution of the location variable λ_A (λ_B, resp.) can be only assigned to the relative address of a single instance of B with respect to one instance of A (of a single instance of A with respect to one instance of B, resp.).*

Proof. By case analysis.

Actually, different instances of the same process are always identified by different instances of location variables. Therefore, two location variables, arising from two different sessions, never point to the same process.

We now define the extension of P to multisession as follows:

$$P^m = m\_startup(\epsilon_\bullet\epsilon, A, \lambda_B, B)$$

Consider now the following execution:

$$
\begin{aligned}
P^m \;&=\; (\nu s)(\ !\overline{s}\langle s\rangle.A \mid !s_{\lambda_B}(x).B\ ) \mid E \\
&\xrightarrow{\tau} (\nu s)(\ (\ A \mid !\overline{s}\langle s\rangle.A\ ) \mid (\ c_{||_0||_0\bullet||_1||_0}(z).B'(z) \mid !s_{\lambda_B}(x).B\ )\ ) \mid E \\
&\xrightarrow{\tau} (\nu s)(\ (\ A \mid (\ A \mid !\overline{s}\langle s\rangle.A\ )\ )\ \mid (\ c_{||_0||_0\bullet||_1||_0}(z).B'(z) \mid \\
&\qquad\quad \mid (\ c_{||_0||_1||_0\bullet||_1||_1||_0}(z).B'(z) \mid !s_{\lambda_A}(x).B\ )\ )\ ) \mid E
\end{aligned}
$$

Here, the first and second instances of B are uniquely hooked to the first and second instances of A, respectively. This implies that all the future located communications of such processes will be performed only with the corresponding hooked partner, even if they are performed on the same communication channel. Generally, due to nondeterminism, instances of A and instances of B may hook in different order.

It is now straightforward proving a couple of properties about authentication and freshness, exploiting Proposition 3. They hold for protocol P^m and for all similar protocols, where multiple sessions arise from the replication of the same processes, playing the same roles. In the following, we use $B'(\vartheta\bullet\vartheta' N)$ to mean the continuation B', where the variable z has been bound to the value $\vartheta\bullet\vartheta' N$, i.e. to a message N that has the relative address $\vartheta\bullet\vartheta'$ of its sender w.r.t. to its receiver.

Authentication: When the continuation of an instance of $B'(\vartheta\bullet\vartheta' N)$ is activated, $\vartheta\bullet\vartheta'$ must be the relative address of an instance of A with respect to the actual instance of B.

Freshness: For every pair of activated instances of continuations $B'(\vartheta\bullet\vartheta' N)$ and $B'(\tilde{\vartheta}\bullet\tilde{\vartheta}' N')$ it must be $\vartheta \neq \tilde{\vartheta}$, i.e., the two messages have been originated by two different instances of the process A.

We are now able to show that P_2 is not a good implementation when many sessions are considered, i.e. that P_2^m does not implement P^m. Consider:

$$P_2^m = (\nu K_{AB})(!A_2 \mid !B_2)$$

Let $B'(z) = \overline{observe}\langle z\rangle$, and consider $E = c(x).\overline{c}\langle x\rangle.\overline{c}\langle x\rangle$. E may intercept the encrypted message and replay it twice. If we consider the tester $T = observe(x).observe(y).[x \overset{@}{=} y]\beta(x)$, we obtain that $(\nu c)(P_2^m \mid E)$ may pass the test (T, β) while $(\nu c)(P^m \mid E)$ never passes it. Indeed, in P_2^m the replay attack successfully performed, and B is accepting twice the same message:

Message $1.a$	$A \to E(B)$:	$\{M\}_{K_{AB}}$	E intercepts the message intended for B
Message $2.a$	$E(A) \to B$:	$\{M\}_{K_{AB}}$	E pretending to be A
Message $2.b$	$E(A) \to B$:	$\{M\}_{K_{AB}}$	E pretending to be A

Thus, we obtain that $(\nu c)(P_2^m \mid E) \not\leq (\nu c)(P^m \mid E)$.

We end this section by giving a correct implementation of the multisession authentication protocol P^m, which exploits a typical challenge-response mechanism to guarantee authentication:

Message 1 $B \to A :\ N$
Message 2 $A \to B :\ \{M, N\}_{K_{AB}}$

where N is a freshly generated nonce that constitutes the challenge. It can be formally specified as follows:

$$\begin{aligned} P_3^m &= (\nu K_{AB})(!A_3 \mid !B_3) \\ A_3 &= (\nu M)c(ns).\overline{c}\langle\{M, ns\}_{K_{AB}}\rangle \\ B_3 &= (\nu N)\overline{c}\langle N\rangle.c(x).case\ x\ of\ \{z, w\}_{K_{AB}}\ in\ [w = N]B'(z) \end{aligned}$$

The following holds.

Proposition 4. *P_3^m securely implements P^m*

Proof. The proof can be carried out in the same style of the one for Proposition 2.

Note that we are only considering protocols in which the roles of the initiator (or sender) and responder (or receiver) are clearly separated. If A and B could play both the two roles in parallel sessions, then the protocol above would suffer of a well-known reflection attack. Extending our technique to such a more general analysis is the object of future research.

References

1. M. Abadi and A. D. Gordon. "A Calculus for Cryptographic Protocols: The Spi Calculus". *Information and Computation*, 148(1):1–70, January 1999.
2. M. Abadi. 'Secrecy by Typing In Security protocols". *Journal of the ACM*, 5(46):18–36, sept 1999.
3. M. Abadi, C. Fournet, G. Gonthier. Authentication Primitives and their compilation. In *Proceedings of Principles of Programming Languages (POPL'00)*, pp. 302–315. ACM Press, 2000.
4. C. Bodei, P. Degano, R. Focardi, and C. Priami. "Primitives for Authentication in Process Algebras". *Theoretical Computer Science* 283(2): 271-304, June 2002.
5. C. Bodei, P. Degano, and C. Priami. "Names of the π-Calculus Agents Handled Locally". *Theoretical Computer Science*, 253(2):155–184, 2001.
6. C. Bodei, P. Degano, R. Focardi, and C. Priami. "Authentication via Localized Names". In *Proceedings of the 12th Computer Security Foundation Workshop (CSFW12)*, pp. 98–110. IEEE press, 1999.
7. M. Boreale and R. De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120(2):279–303, August 1995.
8. M. Burrows, M. Abadi, and R. Needham. "A Logic of Authentication". *ACM Transactions on Computer Systems*, pp. 18–36, February 1990.
9. National Bureau of Standards. "Data Encryption Standard (DES)". FIPS Publication 46, 1977.
10. R. De Nicola and M.C.B. Hennessy. Testing equivalence for processes. *Theoretical Computer Science*, 34:83–133, 1984.
11. A. Durante, R. Focardi, and R. Gorrieri. "A Compiler for Analysing Cryptographic Protocols Using Non-Interference". *ACM Transactions on Software Engineering and Methodology*, vol. 9(4), pp. 488-528, October 2000.
12. P. Degano and C. Priami. "Enhanced Operational Semantics: A Tool for Describing and Analysing Concurrent Systems". To appear in ACM Computing Surveys.

13. P. Degano and C. Priami. "Non Interleaving Semantics for Mobile Processes". *Theoretical Computer Science*, 216:237–270, 1999.
14. F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. "Strand spaces: Why is a security protocol correct?" In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pp. 160–171, 1998. IEEE Press.
15. R. Focardi, R. Gorrieri, and F. Martinelli. "Message authentication through non-interference". In *Proceedings of International Conference in Algebraic Methodology and Software Technology*, LNCS 1816, pp.258-272, 2000.
16. R. Focardi, R. Gorrieri, and F. Martinelli. "Non Interference for the Analysis of Cryptographic Protocols". In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP'00)*, LNCS 1853, Springer, 2000.
17. R. Focardi and F. Martinelli. "A Uniform Approach for the Definition of Security Properties". In *Proceedings of World Congress on Formal Methods in the Development of Computing Systems*, LNCS 1708, pp. 794–813, Springer-Verlag, 1999.
18. R. Focardi, A. Ghelli, and R. Gorrieri. "Using Non Interference for the Analysis of Security Protocols ". In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, DIMACS Center, Rutgers University, 1997.
19. R. Thayer, N. Doraswamy, and R. Glenn. RFC 2411: IP security document roadmap, November 1998.
20. International Organization for Standardization. Information technology - Security techniques - Entity authentication mechanism; Part 1: General model. ISO/IEC 9798-1, Second Edition, September 1991.
21. G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop (CSFW10)*. IEEE press, 1997.
22. G. Lowe. "Breaking and Fixing the Needham-Schroeder Public-key Protocol using FDR". In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, LNCS 1055, pp. 146–166, Springer-Verlag, 1996.
23. R. Kemmerer, C. Meadows, and J. Millen. "Three systems for cryptographic protocol analysis". *J. Cryptology*, 7(2):79–130, 1994.
24. R. Milner, J. Parrow, and D. Walker. "A Calculus of Mobile Processes (I and II)". *Information and Computation*, 100(1):1–77, 1992.
25. J. C. Mitchell, M. Mitchell, and U. Stern. "Automated Analysis of Cryptographic Protocols Using Murϕ". In *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*, pp. 141–153. IEEE Computer Society Press, 1997.
26. Sangiorgi, D. "Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.". PhD Thesis. University of Edinburgh, 1992.
27. S. Schneider. "Verifying authentication protocols in CSP". *IEEE Transactions on Software Engineering*, 24(9), Sept. 1998.
28. B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 1996. Second edition.