Process Algebraic Frameworks for the Specification and Analysis of Cryptographic Protocols ⋆

Roberto Gorrieri[1] and Fabio Martinelli[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy.
[2] Istituto di Informatica e Telematica C.N.R., Pisa, Italy.

Abstract. Two process algebraic approaches for the analysis of cryptographic protocols, namely the spi calculus by Abadi and Gordon and CryptoSPA by Focardi, Gorrieri and Martinelli, are surveyed and compared. We show that the two process algebras have comparable expressive power, by providing an encoding of the former into the latter. We also discuss the relationships among some security properties, i.e., authenticity and secrecy, that have different definitions in the two approaches.

1 Introduction

Security protocols are those protocols that accomplish security goals such as preserving the secrecy of a piece of information during a protocol or establishing the integrity of the transmitted information. Cryptographic protocols are those security protocols running over a public network that use cryptographic primitives (e.g., encryption and digital signatures) to achieve their security goals.

In the analysis of cryptographic protocols, one has to cope with the insecurity of the network. So, it is assumed that one *attacker* (sometimes called *enemy* or *intruder*) of the protocol has complete control over the communication medium. On the other hand, to make analysis less intricate, it is also usually assumed *perfect cryptography*, i.e., such an enemy is not able to perform cryptanalytic attacks: an encrypted message can be decrypted by the enemy only if he knows (or is able to learn) the relevant decryption key. Such an analysis scenario is often referred to as the Dolev-Yao approach [10].

Because of the above, cryptographic protocols are difficult to be analysed and to be proved correct. Indeed, a lot of them have flaws or inaccuracies. As a well-known example, we mention that Lowe [22] pointed out one inaccuracy in an authentication protocol by Needham and Schroeder [31].

Hence, the need of a formal approach to the analysis of cryptographic protocols. The spi calculus [3], proposed by Abadi and Gordon, and CryptoSPA [17,

⋆ Work partially supported by MURST Progetto "Metodi Formali per la Sicurezza" (MEFISTO); IST-FET project "Design Environments for Global ApplicationS (DEGAS)"; Microsoft Research Europe; by CNR project "Tecniche e Strumenti Software per l'analisi della sicurezza delle comunicazioni in applicazioni telematiche di interesse economico e sociale" and by a CSP grant for the project "SeTAPS II".

16], proposed by the authors in joint work with R. Focardi, are two well-known possible answers. The goal of this paper is to show similarities and differencies of these two approaches from the point of view of both modeling and analysis. A small running example is used throughout the paper in order to illustrate the basic features of the two approaches.

The basic idea is that, in order to analyse a protocol, one has to begin by modeling it as a program of the calculus. At first sight, there are some differencies in the spi model and in the CryptoSPA one. In particular, the spi calculus, being based on the π calculus, is apparently more expressive as it handles mobility (of channels) as a first class primitive of the language. On the other hand we show that it is possible to define an encoding from the spi calculus to CryptoSPA that preserves a rather strong notion of equivalence. The core idea of the encoding is that name generation of spi can be simulated by means of a suitable process in CryptoSPA that uses the inference system hidden inside the language. Moreover, the spi calculus offers the possibility to describe secret pieces of information inside the syntax, by means of the restriction operator. On the contrary, in CryptoSPA these secrets are to be specified separately as limitations on the knowedge of the enemy that tries to attack the protocol.

A major difference between the two approaches can be summarised by the motto: *Contextual equivalence vs Equivalence of contexts*. In the spi calculus the properties of secrecy and message authenticity are expressed as the equivalence of systems, where the used notion of equivalence is may testing: it is based on the idea that two systems are equivalent if they cannot be distinguished by an external observer. According to the spi calculus approach, the tester is playing at the same time the role of observer and attacker of the protocol; hence, elegantly, spi includes the notion of external enemy inside the definition of the semantics of the calculus by using a *contextual equivalence*. Indeed, the two processes must exhibit the same observable behavior w.r.t. any context (the observer).

On the contrary, in CryptoSPA the properties of secrecy and authenticity (or integrity) are formulated as instances of the following general form:

$$\forall X \in \mathcal{E}_C^{\phi_I} \quad (S \,|\, X) \setminus C \sim \alpha(S)$$

where X is any process in the set $\mathcal{E}_C^{\phi_I}$ of admissible enemies, C is the set of communication channels, $\sim$ is a behavioural semantics (actually, trace semantics for our purpose) and $\alpha(S)$ is the correct specification of S when run in isolation. The equation above amounts to say that the behaviour of system S when exposed to any enemy X is the same as the correct behaviour of S. Hence, such properties are expressed as a form of *equivalence of contexts*: a *closed* system, i.e. $\alpha(S)$, is compared with an *open* system $(S \,|\, \bullet) \setminus C$ and the comparison takes the form of an infinity of checks between closed systems, for each possible enemy X. (Actually, nothing prevents $\sim$ from being itself a contextual equivalence, though trace equivalence is the usual relation used in the CryptoSPA approach.)

We will show that the latter approach is more flexible, by providing an example of an attack scenario where there are several enemies with different capabilities that can be naturally treated in the CryptoSPA approach. On the contrary, the spi approach is appropriate for modeling a scenario where there is one single enemy, as in the Dolev-Yao approach.

The final part of the paper is devoted to show similarities and differences among secrecy and authenticity as defined in the two approaches. We show that, in spite of the many technical differencies, the notion of spi authenticity is the same as the notion of integrity in CryptoSPA. On the contrary, the two notions of secrecy are quite different.

A short summary of other process algebraic approaches to the analysis of cryptographic protocols concludes the paper.

2 The Spi Calculus

The spi calculus [3] is a version of the π calculus [30] equipped with abstract cryptographic primitives, e.g. primitives for perfect encryption and decryption. Names represent encryption keys as well as communication channels.

Here we give a short overview of the main features of the calculus, by presenting a simple version with asynchronous communication and shared-key cryptography. The interested reader can find more details in [3] and [20] (a tutorial on the subject that has inspired the current short survey).

2.1 Syntax and Reduction Semantics

In this section we briefly recall some basic concepts about the asynchronous spi calculus with shared-key cryptography. The choice of the asynchronous version is inessential for the results of the paper and is only taken for simplicity. The restriction to shared-key cryptography only is for the sake of simplicity too.

Given a countable set of names $\mathcal{N}$ (ranged over by $a, b, \ldots, n, m, \ldots$) and a countable set of variables $\mathcal{V}$ (ranged over by $x, y, \ldots,$), the set of *terms* is defined by the grammar:

$$M, N \ ::= \ m \mid x \mid (M, N) \mid \{M\}_N$$

with the proviso that in (M, N) and $\{M\}_N$ the term M (and similarly N) can be either a *ground* term (i.e., without variable occurrences) or simply a variable.

The set of spi calculus processes is defined by the BNF-like grammar:

$$\begin{aligned} P, Q ::= \mathbf{0} \mid \overline{M}\langle N\rangle \mid M(x).P \mid (\nu n)\, P \mid P \,|\, Q \mid [M = N]P \text{ else } Q \mid A\langle \mathrm{M}_1, \ldots, \mathrm{M}_n\rangle \mid \\ \text{let } (x, y) = M \ \text{ in } P \text{ else } Q \mid \text{case } M \text{ of } \{x\}_N \ \text{ in } P \text{ else } Q \end{aligned}$$

The name n is bound in the term $(\nu n)P$. In $M(x).P$ the variable x is bound in P. In let $(x, y) = M$ in P else Q the variables x and y are bound in P. In case M of $\{x\}_N$ in P else Q the variable x is bound in P. The set $fn(P)$ of free names of P is defined as usual.

We give an intuitive explanation of the operators of the calculus:

- $\mathbf{0}$ is the stuck process that does nothing.
- $\overline{M}\langle N\rangle$ is the output construct. It denotes a communication on the channel M of the term N.

- $M(x).P$ is the input construct. A name is received on the channel M and its value is substituted for the free occurrences of x in P (written $P[M/x]$).
- $(\nu n)P$ is the process that makes a new, private name n, for P, and then behaves as P.
- $P \,|\, Q$ is the parallel composition of two processes P and Q. Each may interact with the other on channels known to both, or with the outside world, independently of the other.
- $[M = N]P$ else Q is the match construct. The process behaves as P when $M = N$, otherwise it behaves as Q.
- $A\langle x_1, \ldots, x_n\rangle$ is a process constant. We assume that constants are equipped by a constant definition like $A\langle x_1, \ldots, x_n\rangle \doteq P$, where the free variables of P are contained in $\{x_1, \ldots, x_n\}$.
- let $(x, y) = M$ in P else Q is the *pair splitting* process. If the term M is of the form (N, L), then it behaves as $P[N/x][L/y]$; otherwise, it behaves as Q.
- case M of $\{x\}_N$ in P else Q is the *decryption* process. If M is of the form $\{L\}_N$, then the process behaves as $P[L/x]$; otherwise, it behaves as Q.

We also define the *structural congruence* as follows. Let $\equiv$ be the least congruence relation over processes closed under the following rules:

1. $P \equiv Q$, if P is obtained through α–conversion from Q
2. $P \,|\, \mathbf{0} \equiv P$
3. $P \,|\, Q \equiv Q \,|\, P$
4. $P \,|(Q \,|\, R) \equiv (P \,|\, Q) \,|\, R$
5. $(\nu n)\mathbf{0} \equiv \mathbf{0}$
6. $(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$
7. $(\nu n)(\nu n)P \equiv (\nu n)P$
8. $(\nu n)\overline{M}\langle N\rangle \equiv \overline{M}\langle N\rangle$, if $n \notin fn(M) \cup fn(N)$
9. $(\nu n)M(x).P \equiv M(x).(\nu n)P$, if $n \notin sort(M)$
10. $(\nu n)(P \,|\, Q) \equiv P \,|(\nu n)Q$ if $n \notin fn(P)$
11. $[M = M']P$ else $Q \equiv P$, if $M = M'$
12. $[M = M']P$ else $Q \equiv Q$, if $M \neq M'$ and M, M' are ground
13. let $(x, y) = (\ M, N)$ in P else $Q \equiv P[M/x][N/y]$
14. let $(x, y) = M$ in P else $Q \equiv Q$, if $M \neq (N, N_1)$, for some N, N_1 and $M \notin \mathcal{V}$
15. case $\{M\}_N$ of $\{x\}_N$ in P else $Q \equiv P[M/x]$
16. case M of $\{x\}_N$ in P else $Q \equiv Q$, if $M \neq \{N'\}_N$, for some N' and $M \notin \mathcal{V}$
17. $A\langle \mathrm{M}_1, \ldots, \mathrm{M}_n\rangle \equiv P[\mathrm{M}_1/x_1, \ldots, \mathrm{M}_1/x_1]$, when $A\langle x_1, \ldots, x_n\rangle \doteq P$.

We give the *reduction* semantics for the asynchronous spi calculus. Processes communicate among them by exchanging messages. An internal communication (or *reduction*) of the process P is denoted by $P \longrightarrow P'$. We have the following rules for calculating the reduction relation between processes:

$$\frac{}{\overline{m}\langle N\rangle \,|\, m(x).P \longrightarrow P[N/x]} \qquad \frac{P \equiv Q, Q \longrightarrow Q', Q' \equiv P'}{P \longrightarrow P'}$$

$$\frac{P \longrightarrow P'}{P \,|\, Q \longrightarrow P' \,|\, Q} \qquad \frac{P \longrightarrow P'}{\nu n(P) \longrightarrow \nu n(P')}$$

2.2 May Testing Semantics

May testing equivalence [9] is the equivalence notion that is used in the spi calculus to define the security properties. In order to define this equivalence, we first define a predicate that describes the channels on which a process can communicate. We let a *barb* β be an output channel. For a closed process P, we define the predicate P *exhibits barb* β, written $P \downarrow \beta$, by the following rules:

$$\frac{}{\overline{m}\langle N\rangle \downarrow \overline{m}} \qquad \frac{P \downarrow \beta}{P \,|\, Q \downarrow \beta}$$

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

Intuitively, $P \downarrow \beta$ holds if P may output immediately along β. The *convergence* predicate $P \Downarrow \beta$ holds if P exhibits β after some reactions.

$$\frac{P \downarrow \beta}{P \Downarrow \beta} \quad \frac{P \longrightarrow Q \quad Q \Downarrow \beta}{P \Downarrow \beta}$$

A *test* consists of any closed process R and any barb β. A closed process P *passes* the test if and only if $(P \,|\, R) \Downarrow \beta$. May testing equivalence is then defined on the set of closed processes as follows:

$$P \approx_{may} Q \iff \text{for any test } (R, \beta), \quad (P \,|\, R) \Downarrow \beta \text{ if and only if } (Q \,|\, R) \Downarrow \beta$$

May-testing has been chosen because it corresponds to partial correctness (or safety), and security properties are often safety properties. Moreover, a test neatly formalises the idea of a generic experiment or observation another process (such as an attacker) might perform on a process. So testing equivalence captures the concept of equivalence in an arbitrary environment; as a matter of fact, may-testing equivalence is a contextual equivalence.

3 CryptoSPA

Cryptographic Security Process Algebra (*CryptoSPA* for short) is a slight modification of CCS process algebra [29], adopted for the description of cryptographic protocols. It makes use of cryptographic-oriented modeling constructs and can deal with confidential values [15, 17, 26].

The CryptoSPA model consists of a set of sequential agents able to communicate by exchanging messages.

The data handling part of the language consists of a set of inference rules used to deduce messages from other messages. We consider a set of relations among closed messages as: $\vdash_r \subseteq \mathcal{P}^{fin}(\mathcal{M}) \times \mathcal{M}$, where r is the name of the rule. Given a set $\mathcal{R}$ of inference rules, we consider the deduction relation $\mathcal{D}^{\mathcal{R}} \subseteq \mathcal{P}^{fin}(\mathcal{M}) \times \mathcal{M}$. Given a finite set of closed messages, say ϕ, then $(\phi, M) \in \mathcal{D}^{\mathcal{R}}$ if M can be derived by iteratively applying the rules in $\mathcal{R}$. For the sake of simplicity, we assume that $\vdash_r$ (for each $r \in \mathcal{R}$) and $\mathcal{D}^{\mathcal{R}} \subseteq \mathcal{P}^{fin}(\mathcal{M}) \times \mathcal{M}$ are decidable.

3.1 The Language Syntax

CryptoSPA syntax is based on the following elements:

- A set Ch of channels, partitioned into a set I of input channels (ranged over by c) and a set O of output channels (ranged over by $\overline{c}$, the output corresponding to the input c);
- A set Var of variables, ranged over by x;
- A set $\mathcal{M}$ of messages, defined as above for the spi calculus, ranged over by M, N (and by m, n, with abuse of notation, to denote *closed* messages).

The set $\mathcal{L}$ of CryptoSPA terms (or processes) is defined as follows:

$$P, Q ::= \mathbf{0} | \; c(x).P \mid \overline{c}M.P \mid \tau.P \mid P \,|\, Q \mid P\backslash L \mid$$

$$A(M_1, \ldots, M_n) \mid [\langle M_1, \ldots, M_r\rangle \vdash_{rule} x]P; Q$$

where $M, M', M_1, \ldots, M_r$ are messages or variables and L is a set of channels. Both the operators $c(x).P$ and $[\langle M_1 \ldots M_r\rangle \vdash_{rule} x]P; Q$ bind variable x in P.

We assume the usual conditions about *closed* and *guarded* processes, as in [29]. We call $\mathcal{P}$ the set of all the *CryptoSPA* closed and guarded terms. The set of actions is $Act = \{c(M) \mid c \in I\} \cup \{\overline{c}M \mid \overline{c} \in O\} \cup \{\tau\}$ (τ is the internal, invisible action), ranged over by a. We define $sort(P)$ to be the set of all the channels syntactically occurring in the term P. Moreover, for the sake of readability, we always omit the termination $\mathbf{0}$ at the end of process specifications, e.g. we write a in place of $a.\mathbf{0}$. We give an informal overview of *CryptoSPA* operators:

- $\mathbf{0}$ is a process that does nothing.
- $c(x).P$ represents the process that can get an input M on channel c behaving like $P[M/x]$).
- $\overline{c}m.P$ is the process that can send m on channel c, and then behaves like P.
- $\tau.P$ is the process that executes the invisible τ and then behaves like P.
- $P_1 \,|\, P_2$ (*parallel*) is the parallel composition of processes that can proceed in an asynchronous way but they must synchronize on complementary actions to make a communication, represented by a τ.
- $P\backslash L$ is the process that cannot send and receive messages on channels in L; for all the other channels, it behaves exactly like P;
- $A(M_1, \ldots, M_n)$ behaves like the respective defining term P where all the variables $x_1, \ldots, x_n$ are replaced by the messages $M_1, \ldots, M_n$;
- $[\langle M_1, \ldots, M_r\rangle \vdash_{rule} x]P; Q$ is the process used to model message manipulation as cryptographic operations. Indeed, the process $[\langle M_1, \ldots, M_r\rangle \vdash_{rule} x]P; Q$ tries to deduce an information z from the tuple $\langle M_1, \ldots, M_r\rangle$ through the application of rule $\vdash_{rule}$; if it succeeds then it behaves like $P[z/x]$, otherwise it behaves as Q. The set of rules that can be applied is defined through an inference system (e.g., see Figure 1 for an instance).

$$\frac{m \quad m'}{(m,m')}(\vdash_{pair}) \quad \frac{(m,m')}{m}(\vdash_{fst}) \quad \frac{(m,m')}{m'}(\vdash_{snd})$$

$$\frac{m \quad k}{\{m\}_k}(\vdash_{enc}) \qquad \frac{\{m\}_k \quad k}{m}(\vdash_{dec})$$

Fig. 1. An example inference system for shared key cryptography

3.2 The Operational Semantics of CryptoSPA

In order to model message handling and cryptography we use a set of inference rules. Note that *CryptoSPA* syntax, its semantics and the results obtained are completely parametric with respect to the inference system used. We present in Figure 1 an instance inference system, with rules: to combine two messages obtaining a pair (rule $\vdash_{pair}$); to extract one message from a pair (rules $\vdash_{fst}$ and $\vdash_{snd}$); to encrypt a message m with a key k obtaining $\{m\}_k$ and, finally, to decrypt a message of the form $\{m\}_k$ only if it has the same key k (rules $\vdash_{enc}$ and $\vdash_{dec}$, respectively).

In a similar way, inference systems can contain rules for handling the basic arithmetic operations and boolean relations among numbers, so that the value-passing CCS **if-then-else** construct can be obtained via the $\vdash_{rule}$ operator.

Example 1. Natural numbers may be encoded by assuming a single value 0 and a function $S(y)$, with the following rule: $\frac{x}{S(x)}$ *inc*. Similarly, we can define summations and other operations on natural numbers. ■

Example 2. We do not explicitly define equality check among messages in the syntax. However, this can be implemented through the usage of the inference construct. E.g., consider rule $\frac{x \quad x}{Equal(x,x)}$ *equal*. Then $[m = m']A$ (with the expected semantics) may be equivalently expressed as $[m \quad m' \vdash_{equal} y]A$ where y does not occur in A. Similarly, we can define inequalities, e.g., $\leq$, among natural numbers. ■

More interestingly, this form of inference constructs of CryptoSPA is also useful to model common access control mechanisms in distributed systems.

Example 3. Indeed, consider a set of credentials, i.e. (signed) messages containing information about access rights. Assume that $\{A, ob_1, +\}_{pr(C)}$ means that the user C (via the signature with its private key $pr(C)$) asserts A has the right to access the object ob_1 and may grant this access to other users (this is denoted through the symbol $+$). A rule like:

$$\frac{\{A, ob_1, +\}_{pr(C)} \quad pr(C) \quad \{grant \quad B, ob_1\}_{pr(A)}}{\{B, ob_1, +\}_{pr(C)}}(acc_C)$$

may be used by the controller C to issue other access right credentials, after receiving an indication by A, i.e. the signed message $\{grant \quad B, ob_1\}_{pr(A)}$.

$$(input)\frac{m \in \mathcal{M}}{c(x).P \xrightarrow{c(m)} P[m/x]} \qquad (output)\frac{}{\overline{c}m.P \xrightarrow{\overline{c}m} P} \qquad (internal)\frac{}{\tau.P \xrightarrow{\tau} P}$$

$$(\backslash L)\frac{P \xrightarrow{c(m)} P' \quad c \notin L}{P\backslash L \xrightarrow{c(m)} P'\backslash L} \qquad (|)_1\frac{P_1 \xrightarrow{a} P_1'}{P_1 \,|\, P_2 \xrightarrow{a} P_1' \,|\, P_2} \qquad (|)_2\frac{P_1 \xrightarrow{c(x)} P_1' \quad P_2 \xrightarrow{\overline{c}m} P_2'}{P_1 \,|\, P_2 \xrightarrow{\tau} P_1' \,|\, P_2'}$$

$$(Def)\frac{P[m_1/x_1, \ldots, m_n/x_n] \xrightarrow{a} P' \quad A(x_1, \ldots, x_n) \doteq P}{A(m_1, \ldots, m_n) \xrightarrow{a} P'}$$

$$(\mathcal{D})\frac{\langle m_1, \ldots, m_r \rangle \vdash_{rule} m \quad P[m/x] \xrightarrow{a} P'}{[\langle m_1, \ldots, m_r \rangle \vdash_{rule} x]P; Q \xrightarrow{a} P'}$$

$$(\mathcal{D}_1)\frac{\nexists m \text{ s.t. } \langle m_1, \ldots, m_r \rangle \vdash_{rule} m \quad Q \xrightarrow{a} Q'}{[\langle m_1, \ldots, m_r \rangle \vdash_{rule} x]P; Q \xrightarrow{a} Q'}$$

Fig. 2. Structured Operational Semantics for CryptoSPA (symmetric rules for $|_1, |_2$ and $\backslash L$ are omitted)

Thus, we may also consider the inference rules as an abstract mechanism to express security policies usually defined using other mathematical models and logics (e.g., see [21, 34]).

The operational semantics of a $CryptoSPA$ term is described by means of the *labelled transition system* (*lts*, for short) $\langle \mathcal{P}, Act, \{\xrightarrow{a}\}_{a \in Act}\rangle$, where $\{\xrightarrow{a}\}_{a \in Act}$ is the least relation between $CryptoSPA$ processes induced by the axioms and inference rules of Figure 2. As a notation we also use $P \Longrightarrow P'$ for denoting that P and P' belong to the reflexive and transitive closure of $\xrightarrow{\tau}$; $P \overset{\gamma}{\Longrightarrow} P'$ if γ is a finite sequence of actions $a_i, 1 \leq i \leq n$ s.t. $a_i \neq \tau$ and $P \overset{\tau}{\Longrightarrow}\xrightarrow{a_1}\Longrightarrow \ldots \Longrightarrow\xrightarrow{a_n}\overset{\tau}{\Longrightarrow} P'$.

Let $Tr(P)$ denote the set $\{\gamma \in (Act \setminus \{\tau\})^* | P \overset{\gamma}{\Longrightarrow} P'\}$ of executable observable traces. We define the trace preorder, $\leq_{trace}$, as follows: $P \leq_{trace} Q$ if $Tr(P) \subseteq Tr(Q)$. We say that P and Q are *trace equivalent*, denoted $P \sim_{tr} Q$, iff $Tr(P) = Tr(Q)$.

4 Comparison in Expressiveness

We compare the two languages by providing an encoding from the spi calculus to CryptoSPA (actually a sublanguage). Basically, the encoding $[\ \ ]$, preserves the equality of terms, i.e. assume that $\approx_1$ ($\approx_2$) is an equivalence relation on spi calculus (CryptoSPA), then

$$P \approx_1 Q \quad \Longleftrightarrow \quad [P] \approx_2 [Q]$$

The technical machinery in [35] about barbed equivalences will be of help, as $\approx_i$, with $i = 1, 2$, will be weak barbed equivalences. Barbed equivalences are based on a minimal notion of observable, i.e. the barb. This makes them very suitable to provide natural equivalence notions in different languages in a uniform way. It is worthwhile noticing that these forms of equivalence are finer than may-testing (in their respective languages).

Barbed equivalences Given the predicate P *exhibits a barb* β, $P \downarrow \beta$, it is possible to define in a standard way a set of useful process equivalences. We say that a symmetric relation $\mathcal{R}$ among processes is a barbed bisimulation, if $(P, Q) \in \mathcal{R}$ then:

- For all β, it holds $P \downarrow \beta$ iff $Q \downarrow \beta$;
- if $P \longrightarrow P'$ then $\exists\ Q'$ s.t. $Q \longrightarrow Q'$ and $(P', Q') \in \mathcal{R}$.

The union of all barbed bisimulations, denoted by $\sim$, is a barbed bisimulation. There exists also a form of weak barbed bisimulation where in the previous statements, the observable predicate is replaced by the weak one, i.e. $P \Downarrow \beta$ and $Q \longrightarrow Q'$ by $Q \Longrightarrow Q'$. We say that a symmetric relation $\mathcal{R}$ among processes is a barbed equivalence $\simeq$ whenever, if $(P, Q) \in \mathcal{R}$ then for each static (*cf* [35]) context $C[\cdot]$, it holds that $C[P] \sim C[Q]$.

Our encoding works in two steps:

- The spi calculus will be encoded, up to weak barbed equivalence, into a sublanguage, called spi_{res}-calculus. In this sublanguage, the input operator is replaced by a new one: a process can receive only pairs on a public channel, i.e. *net*, that cannot be restricted. After receiving a pair, a process is obliged to check the first element of the pair with a given message. Only if the match is successful the process proceeds, otherwise it has to reproduce the message in the net and to try with another pair.
- Then, the spi_{res}-calculus is encoded into CryptoSPA$_{res}$, a similar variant of spi_{res} but on CryptoSPA. Basically, we encode the decryption, splitting and matching constructs through inference rules. Moreover, the new name generation is simulated as the receiving action of a fresh message generated by a special process, called *Gen*.

4.1 An Encoding of spi Calculus into spi_{res}-Calculus

The encoding $[P]_1$ acts as an homomorphism on spi calculus process, except for $\overline{\mathrm{M}}\langle\mathrm{N}\rangle.P$ and $\mathrm{M}(x).P$. In particular, we have the homomorphic $[(\nu c)P]_1 = (\nu c)[P]_1$; moreover, $[M\langle N\rangle]_1 = net\langle(M, N)\rangle$, where *net* is a special channel name that cannot be restricted, and $[M(x).P]_1 = A$ where the defining equation for A is

$$\begin{aligned} A \doteq\ & net(x).\text{let } (z_1, z_2) = x \\ & \quad \text{in } ([M = z_1][P[z_2/x]]_1 \text{ else } \overline{net}\langle(z_1, z_2)\rangle \mid A) \\ & \quad \text{else } \overline{net}\langle x\rangle \mid A \end{aligned}$$

The encoding works as follows. Sent messages are encoded as pairs: the first element denotes the channel, and the second one the message itself. When a process wishes to receive a message on a certain channel, say M, it has to get a pair from the network, and then it is obliged to check if the first element of the pair is the channel; if so, the process proceeds as before (provided that the derivative is encoded), otherwise the pair that has been captured from the network is inserted again. Note that we cannot avoid that a communication happens within the channel net, however, in the case that the channel is not the expected one then the system returns to the original configuration. Consider the following example of a communication on a restricted channel.

Example 4. Suppose $P = \nu c(\overline{c}\langle n\rangle \mid c(x).\overline{x}\langle x\rangle)$. Then, $[P]_1$ is $\nu c(\overline{net}\langle(c,n)\rangle \mid A)$ where A is defined as

$$\begin{aligned} A \doteq\ & net(x).\text{let } (z_1,z_2)=x \quad \text{in } ([c=z_1](\overline{net}\langle(z_2,z_2)\rangle) \text{ else } \overline{net}\langle(z_1,z_2)\rangle \mid A) \\ & \text{else } \overline{net}\langle x\rangle \mid A \end{aligned}$$

Now, $P \longrightarrow \nu c(\overline{n}\langle n\rangle)$ and similarly $[P]_1 \longrightarrow \nu c(\overline{net}(n,n))$. Consider now the process $Q = \overline{c_1}\langle n\rangle$. Then, $[P \mid Q]_1 = [P]_1 \,|[Q]_1 = \nu c(\overline{net}(c,n) \mid A) \mid \overline{net}\langle c_1,n\rangle$. Note that $[P \mid Q]_1 \longrightarrow T$, by means of a synchronization of net and $T \equiv [P \mid Q]_1$:

$$\begin{aligned} & [P \mid Q]_1 \\ = \ & \nu c(\overline{net}\langle(c,n)\rangle \mid A) \mid \overline{net}\langle(c_1,n)\rangle \\ \equiv \ & \nu c(\overline{net}\langle(c,n)\rangle \mid A \mid \overline{net}\langle(c_1,n)\rangle) \\ \longrightarrow \ & \nu c(\overline{net}\langle(c,n)\rangle \,|([c=c_1](\overline{net}\langle(n,n)\rangle) \text{ else } \overline{net}\langle(c_1,n)\rangle \mid A)) \\ \equiv \ & \nu c(\overline{net}\langle(c,n)\rangle \mid A \mid \overline{net}\langle(c_1,n)\rangle) \end{aligned}$$

Indeed, the encoded process may perform useless communications on the channel net; the crucial point is that such communications do not significantly change the status of the process.

Thus, we may define a form of weak barbed equivalence among processes in spi and the ones obtained through $[\quad]_1$, i.e. spi_{res}. The idea is that whenever P exhibits an output on a barb c, say $P \downarrow c$, then $[P]_1$ exhibits an output on a barb net of c, say $[P]_1 \downarrow \overline{net}\langle c,*\rangle$, and conversely. Moreover, if P performs a reduction then also $[P]_1$ must perform it, on the contrary if $[P]_1$ performs a reduction P may also choose to stay blocked. This encoding is clearly not satisfactory for the point of view of implementation, as it introduces divergence. However, it is useful when we are simply interested in verifying security properties that usually depends on may testing equivalence. Indeed, weak barbed equivalence implies may testing equivalence.

4.2 Encoding spi_{res} into CryptoSPA$_{res}$

We may encode the spi_{res} calculus into CryptoSPA$_{res}$. For most operators, the $[\quad]_2$ function works as a homomorphism, e.g., $[P \mid Q]_2 = [P]_2 \,|[Q]_2$. However, we must face two relevant problems: (i) the different forms of cryptography handling;(ii) the different treatment of new name generation.

We deal first with the simpler one that is the treatment of cryptography. In CryptoSPA cryptographic primitives are modeled by means of the inference system. Hence, we can map the decryption construct of spi as follows:

$$[\text{case } M \text{ of } \{x\}_N \text{ in } P \text{ else } Q]_2 = [M \quad N \vdash_{dec} x][P]_2;[Q]_2$$

and similarly for the splitting construct and the matching one.

For the second problem, we have to consider the restriction operator (new name generation). The restriction operator of the spi calculus is used to denote "secret" values known locally by the process. The treatment of such secret values is very elegant in the pi/spi calculus. Ultimately, the operator $(\nu n)P$ defines a new fresh name n in P, that no one else should be ever able to create/guess. We encode these features through the usage of a specific process that creates new names. This process is the unique one allowed to generate such messages and it creates them iteratively. A CryptoSPA specification for such a process could be the following:

$$Gen(x) = [x \vdash_{nonce} y].\overline{gen}y.Gen(y)$$

where the (omitted) rule for nonce creation could be as *inc* in Example 1. The encoding $[\ \ ]_2$ may map each $\nu n(P)$ construct to a receiving action, i.e. $[\nu n(P)]_2 = gen(x).[P]_2$ This second encoding $[\ \ ]^*$ will be actually the following $[P]^* = [P]_2 \,|\, Gen(\mathbf{0})$, because we need one single instance of the name generator process. Note that restricted names are mapped to new names that are unguessable by the enemy because we assume that only process *Gen* can send along *gen*, hence ensuring that an enemy cannot eavesdrop new names sent from *Gen*.

The encoding $[\ \ ]^*$ is sound as we consider as observable the first component in a pair which is an output over *net*, taking care not to consider in CryptoSPA$_{res}$ the new (nonce) names (which correspond to restricted channels in spi_{res}). We show a complete example of encoding from *spi* to CryptoSPA$_{res}$.

Example 5. Consider $P = \nu c(\overline{c}\langle n\rangle)$. Then, $[P]^* = gen(c).\overline{net}(c, n) \,|\, Gen(\mathbf{0})$. Note that although $[P]^*$ may perform a communication step, that cannot be matched by P, the observable behaviour is the same since the output of nonce values cannot be observed.

5 Comparison of the Two Approaches

5.1 Protocol Analysis in the spi Calculus

We show a very basic example. We have two principals A and B that use a public (hence insecure) channel, c_{AB}, for communication; in order to achieve privacy, messages are encrypted with a shared key K_{AB}. The protocol is simply that A sends along c_{AB} a single message M to B, encrypted with K_{AB}.

$$A \to B : \{M\}_{K_{AB}} \text{ on public } c_{AB}$$

The spi calculus specification is as follows:

$$\begin{aligned}
A(M) &= \overline{c_{AB}}\langle \{M\}_{K_{AB}} \rangle \\
B &= c_{AB}(x).\text{case } x \text{ of } \{y\}_{K_{AB}} \text{ in } F(y) \\
P(M) &= (\nu K_{AB})(A(M) \,|\, B)
\end{aligned}$$

where $F(y)$ is the continuation of B. The fact that the channel c_{AB} is public is witnessed by the fact that it is not restricted. On the other hand, K_{AB} is restricted to model that it is a secret known only by A and B. When B receives $\{M\}_{K_{AB}}$ on c_{AB}, B attempts to decrypt it using K_{AB}; if this decryption succeeds, B applies F to the result. Two important properties hold for this protocol:

– Authenticity (or integrity): B always applies F to the message M that A sends; an enemy cannot cause B to apply F to some other message M'.
– Secrecy: No information on the message M can be inferred by an observer while M is in transit from A to B: if F does not reveal M, then the whole protocol does not reveal M.

Intuitively, the secrecy property should establish that if $F(M)$ is indistinguishable from $F(M')$, then the protocol with message M is indistinguishable from the protocol with message M'. This intuition can be formulated in terms of equivalences as follows: if $F(M) \approx_{may} F(M')$, for any M, M', then $P(M) \approx_{may} P(M')$.

Also integrity can be formalized in terms of an equivalence. This equivalence compares the protocol with another version of the protocol which is secure by construction. For this example, the required specification is:

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

The principal B is replaced with a variant $B_{\text{spec}}(M)$ that receives an input from A and then acts like B when B receives M. $B_{\text{spec}}(M)$ is a sort of "magical" version of B that knows the message M sent by A, hence ensuring integrity by construction. Therefore, we take the following equivalence as our integrity property: $P(M) \approx_{may} P_{\text{spec}}(M)$, for any M.

5.2 Protocol Analysis in CryptoSPA

In this section we want to show how to use CryptoSPA for the analysis of cryptographic protocols. The following subsections are devoted (i) to illustrate how security properties can be specified by decorating suitably protocol specification, then (ii) to discuss the actual definition of admissible attackers and, finally, (iii) to show how secrecy and integrity can be modeled in this framework.

Noninterference for Cryptographic Protocols Analysis Noninterference essentially says that a system P is secure if its low behaviour in isolation is the same as its low behaviour when exposed to the interaction with any high level process Π. Analogously, we may think that a protocol P is secure if its (low) behaviour is the same as its (low) behaviour when exposed to the possible attacks of any intruder X.

To set up the correspondence, this analogy forces to consider the enemies as the high processes. Since the enemy has complete control over the communication medium, the CryptoSPA *public* channels in set C (i.e., the names used for message exchange) are the high level actions while the *private channels* in set $(I \cup O) \setminus C$ are the low level ones. As a protocol specification is usually completely given by message exchanges, it may be not obvious what are the low level actions. In our approach, they are extra observable actions that are included into the protocol specification to observe properties of the protocol. Of course, the choice of these extra actions (and the place into the specification where they are to be inserted) is property dependent.

Considering the example

$$A \to B : \{M\}_{K_{AB}} \text{ on public } c_{AB}$$

the basic integrity property that we want to model can be obtained by enriching the protocol specification with a (low) extra action $\overline{received}(M)$ that B performs when receiving the message M. Hence, the protocol specification is:

$$\begin{aligned} A(M) &= \overline{c_{AB}}\{M\}_{K_{AB}} \\ B &= c_{AB}(x).[\langle \{M\}_{K_{AB}}, K_{AB} \rangle \vdash_{dec} y].\overline{received}y \\ P(M) &= A(M) \,|\, B \end{aligned}$$

where $\overline{c_{AB}}\{M\}_{K_{AB}}$ is a shorthand for $[\langle M, K_{AB} \rangle \vdash_{enc} x]\overline{c_{AB}}x$ and where the received message is sent along the private channel *received*.

Hence, we can state that integrity holds if the following equation holds for the protocol enriched with the event $\overline{received}$:

$P(M)$ satisfies integrity of M if for all (admissible) enemies X we have

$$P(M) \setminus \{c_{AB}\} \sim_{tr} (P(M) \,|\, X) \setminus \{c_{AB}\}$$

This noninterference-based definition seems intuitively quite strong; it is of the form of *equivalence of contexts*: the closed term defining the security property, i.e. $P(M) \setminus \{c_{AB}\} \sim_{tr} \{\overline{received}(M)\}$, is checked for (trace) equivalence with the open system $(P(M) \,|\, \bullet) \setminus \{c_{AB}\}$; such a comparison takes the form of an infinity of equivalence checks for all possible enemies X closing the term.

Admissible Enemies Intuitively, an enemy X can be thought of as a process which tries to attack a protocol by stealing and faking the information which is transmitted on the CryptoSPA public channels in set C. However, we are to be sure that X is not a too powerful attacker. Indeed, a peculiar feature of the enemies is that they should not be allowed to know secret information in advance: as we assume perfect cryptography, the initial knowledge of an enemy must be limited to include only publicly available pieces of information, such as names of entities and public keys, and its own private data (e.g., enemy's private key). If we do not impose such a limitation, the attacker would be able to "guess" every secret piece of information.

Considering the example above, if the enemy knows the key k_{AB} that should be known only by A and B, the protocol would be easily attacked by the instance of the enemy

$$X(m, k) \stackrel{\text{def}}{=} \overline{c_{AB}}\{m\}_k$$

where $m = M_X$ and $k = K_{AB}$. The problem of guessing secret values can be solved by imposing some constraints on the initial data known by the enemies. Given a process P, we call $ID(P)$ the set of messages that occur syntactically in P. Now, let $\phi_I \subseteq \mathcal{M}$ be the *finite*, initial knowledge that we would like to give to the enemies, i.e., the public information such as the names of the entities and the public keys, plus some possible private data of the intruders (e.g., their private keys or nonces). For a certain intruder X, we want that all the messages in $ID(X)$ are deducible from ϕ_I. Formally, given a finite set $\phi_I \subseteq \mathcal{M}$, called the *initial knowledge*, we define the set $\mathcal{E}_C^{\phi_I}$ of *admissible enemies* as $\mathcal{E}_C^{\phi_I} = \{X \in \mathcal{P} \mid sort(X) \subseteq C \text{ and } ID(X) \subseteq \mathcal{D}(\phi_I)\}$. To see how $\mathcal{E}_C^{\phi_I}$ prevents the problem presented in the running example, to indicate that K_{AB} is secret, we can now require that $K_{AB} \notin \mathcal{D}(\phi_I)$. Since $ID(X(M_X, K_{AB})) = \{M_X, K_{AB}\}$, we finally have that $X(M_X, K_{AB}) \notin \mathcal{E}_C^{\phi_I}$.

Integrity In order to specify integrity (or authenticity), one has simply to decorate the protocol specification P with an action of type *received* in correspondence of the relevant point of the protocol specification, obtaining a decorated protocol P'. Once this has been done, the integrity equation reads as follows:

$$P(M) \text{ satisfies integrity of } M \text{ if } \forall X \in \mathcal{E}_C^{\phi_I} (P'(M) \mid X) \setminus C \sim_{tr} P'(M) \setminus C$$

where C is the set of public channels. The intuition is that $P'(M) \setminus C$ represents the protocol P running in isolation (because of the restriction on public channels), while $(P'(M) \mid X) \setminus C$ represents the protocol under the attack of an admissible (i.e., that does not know too much) enemy X. The equality imposes that the enemy X is not able to violate the integrity property specification that is represented by the correct (low) trace $\overline{received}(M)$.

The property above is very similar to a property known in the literature as *Non Deducibility on Compositions* [13, 14] (NDC for short) and, as we will show in the next subsection is valid also for analysing secrecy. NDC for CryptoSPA is defined as follows (see [17]): A process S is NDC iff $\forall X \in \mathcal{E}_C^{\phi_I} \quad (S \mid X) \setminus C \sim_{tr} S \setminus C$. In other words S is NDC if every possible enemy X which has an initial knowledge limited by ϕ_I is not able to significantly change the behaviour of the system. This definition can be generalized to the scheme GNDC [15, 17] as follows:

$$S \text{ is } GNDC^{\alpha}_{\approx} \text{ iff } \forall X \in \mathcal{E}_C^{\phi_I} : (S \mid X) \setminus C \approx \alpha(S)$$

where $\alpha(S)$ denotes the secure specification of the system, which is then compared with the open term $(S \mid \bullet) \setminus C$. GNDC is a very general scheme under which many security properties for cryptographic protocols can be defined as suitable instances. See, e.g., [16] for some examples about authentication properties.

Secrecy Also secrecy can be defined via a variation of the NDC equation above, or better as an instance of the GNDC scheme. Consider a protocol $P(M)$ and assume that we want to verify if $P(M)$ preserves the secrecy of message M.

This can be done by proving that every enemy which does not know message M, cannot learn it by interacting with $P(M)$. Thus, we need a mechanism that notifies whenever an enemy is learning M. We implement it through a simple process called *knowledge notifier* which reads from a public channel $c_k \in C \setminus sort(P(M))$ not used in $P(M)$ and executes a $\overline{learnt}M$ action if the read value is exactly equal to M. For a generic message m, it can be defined as follows:

$$KN(m) \stackrel{\text{def}}{=} c_k(y).[m = y]\overline{learnt}m$$

We assume that *learnt* is a special channel that is never used by protocols and is not public, i.e., $learnt \notin sort(P) \cup C$. We now consider $P'(M) \stackrel{\text{def}}{=} P(M) \,|\, KN(M)$, i.e., a modified protocol where the learning of M is now notified. A very intuitive definition of secrecy can be thus given as follows:

$$\begin{array}{l} P(M) \text{ preserves the secrecy of } M \text{ iff} \\ \quad \text{for all secrets } N \in \mathcal{M} \setminus \mathcal{D}(\phi_I) \quad \forall X \in \mathcal{E}_C^{\phi_I} \quad P'(N) \setminus C \sim_{tr} (P'(N) \,|\, X) \setminus C \end{array}$$

In other words, we require that for every secret M and for every admissible enemy X, process $(P'(M) \,|\, X) \setminus C$ never executes a $\overline{learnt}M$ action, as $P'(N) \setminus C$ is not able to do so.

Most powerful enemy A serious obstacle to the widespread use of these *GNDC*-like properties is the universal quantification over all admissible enemies. While the proof that a protocol is not *NDC* can be naturally given by exhibiting an enemy that breaks the semantic equality, much harder is the proof that a protocol is indeed *NDC*, as it requires an infinity of equivalence checks, one for each admissible enemy. One reasonable way out could be to study if there is an attacker that is more powerful than all the others, so that one can reduce the infinity of checks to just one, albeit huge, check with respect to such most powerful enemy.

Indeed, it is easy to prove that if $\lhd$ is a pre-congruence [3] and if there exists a process $Top \in \mathcal{E}_C^{\phi_I}$ such that for every process $X \in \mathcal{E}_C^{\phi_I}$ we have $X \lhd Top$, then:

$$P \in NDC_{\lhd} \qquad \text{iff} \qquad (P \,|\, Top) \setminus C \;\lhd\; P \setminus C$$

If the hypotheses of the proposition above hold, then it is sufficient to check that $P \setminus C$ is equivalent to $(P \,|\, Top) \setminus C$.

Given the pre-congruence $\lhd$, let $\approx = \lhd \cap \lhd^{-1}$. If there exist two processes $Bot, Top \in \mathcal{E}_C^{\phi_I}$ such that for every process $X \in \mathcal{E}_C^{\phi_I}$ we have $Bot \lhd X \lhd Top$ then

$$P \in NDC_{\approx} \quad \text{iff} \quad (P \,|\, Bot) \setminus C \approx (P \,|\, Top) \setminus C \approx P \setminus C$$

Given these very general results, one may wonder if they are instanciable to some of the semantics we have described so far. Indeed, this is the case, at least for the trace preorder $\leq_{trace}$, which is a pre-congruence.

[3] A preorder $\lhd$ is a *pre-congruence* (w.r.t. the operators $|$ and $\setminus C$) if for every $P, Q, R \in \mathcal{P}$ if $Q \lhd R$ then $P \,|\, Q \lhd P \,|\, R$ and $Q \setminus C \lhd R \setminus C$.

The easy part is to identify the minimal element *Bot* in $\mathcal{E}_C^{\phi_I}$ w.r.t. $\leq_{trace}$: the minimum set of traces is the emptyset, that is generated, e.g., by process $\mathbf{0}$.

Let us now try to identify the top element *Top* in $\mathcal{E}_C^{\phi_I}$ w.r.t. $\leq_{trace}$. The "most powerful enemy" can be defined by using a family of processes $Top_{trace}^{C,\phi}$ each representing the instance of the enemy with knowledge ϕ:

$$Top_{trace}^{C,\phi} = \sum_{\substack{c \in C \\ m \in Msg(c)}} c(m).Top_{trace}^{C,\phi\cup\{m\}} + \sum_{\substack{c \in C \\ m \in \mathcal{D}(\phi) \cap Msg(c)}} \overline{c}m.Top_{trace}^{C,\phi}$$

The "initial element" of the family is Top_{trace}^{C,ϕ_I} as ϕ_I is the initial knowledge. Note that it may accept any input message, to be bound to the variable x which is then added to the knowledge set $\phi \cup \{x\}$, and may output only messages that can transit on the channel c and that are deducible from the current knowledge set ϕ via the deduction function $\mathcal{D}$. It is easy to see that Top_{trace}^{C,ϕ_I} is the top element of the trace preorder. As a consequence of the fact that the trace preorder is a pre-congruence and that Top_{trace}^{C,ϕ_I} is the top element for that preorder, we have that the single check against the top element is enough to ensure *NDC*. Formally: $P \in NDC_C^{\phi_I}$ iff $(P \,|\, Top_{trace}^{C,\phi_I}) \setminus C \sim_{tr} P \setminus C$.

The Example The example studied for the spi calculus can be easily modeled in CryptoSPA:

$$\begin{aligned} A(M) &= \overline{c_{AB}}\{M\}_{K_{AB}} \\ B &= c_{AB}(x).[\langle \{M\}_{K_{AB}}, K_{AB}\rangle \vdash_{dec} y]\mathbf{0} \\ P'(M) &= A(M) \,|\, B \end{aligned}$$

In order to study if integrity and secrecy hold, we have to define a suitably decorated version P' of the protocol P:

$$\begin{aligned} A(M) &= \overline{c_{AB}}\{M\}_{K_{AB}} \\ B &= c_{AB}(x).[\langle \{M\}_{K_{AB}}, K_{AB}\rangle \vdash_{dec} y]\overline{received}y \\ P'(M) &= A(M) \,|\, B \,|\, KN(M) \end{aligned}$$

where we have inserted the extra event $\overline{received}$ for integrity analysis purpose and the knowledge notifier for secrecy analysis purpose. In the single session described above, both secrecy and integrity holds, that is $(P'(M) \,|\, X) \setminus C$ can never show up a low trace that is not possible for $P'(M) \setminus C$.

5.3 Comparing the Analysis Scenarios

In the spi calculus, the analysis scenario is rather delicate: a tester is at the same time the enemy that tries to influence the behaviour of the system and the observer that should keep track of the behaviour of the system. On the contrary, in CryptoSPA the two roles are separate: on the one hand we explicitly introduce the enemy X inside the scope of restriction, on the other hand we use trace equivalence to compare the two behaviours. In essence, we can say that in spi we use a contextual equivalence, while in CryptoSPA we use and equivalence of contexts. We argue that the elegance of the spi approach is paid in terms of lack of flexibility in modeling enemies; for instance, it is not straightforward to model passive attackers (less powerful enemies) or situations in which different parties of the protocol are subject to different enemies (more holes in the context). Both cases can be easily represented in the CryptoSPA approach by choosing either suitable $\mathcal{E}_C^{\phi_I}$ or by including more enemies inside the GNDC-like equations.

Example 6. Consider a system with three components, say A, B and C. The components A and B can communicate on a private channel c and share a secret b, while the components B and C through the private channel c_1 and shares a secret d. Assume that both A and C are malicious; this means to regard them as enemies. Considering a generic specification for A (resp. C), as X_A (resp. X_C), then a possible context to be analyzed is $\nu c \nu b(X_A \,|\, \nu c_1 \nu d(B \,|\, X_C))$, with $b, c \notin fn(X_C)$[4]. As an extension of the spi-calculus approach, we may define a suitable class of contexts w.r.t. one performs certain observations. Thus the previous analysis problem could be instantiated as a contextual equivalence problem. However, techniques for dealing with such generic contexts are currently not well developed. On the contrary, within the GNDC approach, a simple generalization to having more “holes” (unknown components as remarked in [25, 26]), makes it possible to model and analyze it in the case of trace equivalence by resorting to the use of the most general enemies[5]. As a matter of fact, assume that $\mathcal{E}_c^{\phi_A}$ (resp. $\mathcal{E}_{c_1}^{\phi_C}$) denotes the possible behaviours of enemies with the knowledge of ϕ_A (resp. ϕ_C). Then, the GNDC specification could be

$$((X_A) \,|\, B \,|(X_C)) \setminus \{c, c_1\} \sim_{tr} \alpha(B)$$

Note that $\alpha(B)$ could also take into account the description of A and C. Using the most general intruder approach the verification is equal to just one check. As a matter of fact we may simply check that:

$$((Top_c^{\phi_A}) \,|\, B \,|(Top_{c_1}^{\phi_C})) \setminus \{c, c_1\} \sim_{tr} \alpha(B)$$

A main difference, w.r.t. the two approaches may be noted in the treatment of integrity. In the spi-calculus approach we must find the magical correct implementation to be used as reference w.r.t. the system under investigation. (We use the term implementation because it is very close to the description of the system). On the contrary, in the GNDC approach, we simply specify the intended observable behavior, indeed, that the messages are correctly delivered through a control action. Thus, the correct specification is indeed rather more abstract

4 Due to the limitations of using a spi term for expressing the system, it seems difficult to specify such a case study without imposing some side conditions on the processes X_A and X_C.

5 In several cases, to have more enemies that have the possibility to directly communicate with each other is equivalent to consider just one enemy (see [26]).

than the system. Recently Gordon and Jeffrey (e.g., see [19]) developed type systems for a spi-calculus variant that embodies a form of control (correspondence) actions. Using that type systems they were able to check authentication properties as agreement ([33]). In that framework, authenticity is exactly specified as control actions, following the Woo-Lam approach (see [36]).

Another difference is that in spi it is necessary to perform two different analyses in order to prove the two security properties of secrecy and authenticity. On the contrary, in CryptoSPA one single NDC check is enough for both, as both properties are in the NDC form. As a matter of fact, it is enough to consider the decorated specification which includes the control action for integrity and the knowledge notifier. Then, we can use this single, combined specification for a single check against the most powerful enemy. This idea of combined analysis can be generalized to many different properties and has shown its usefulness (higher probability to find unexplored attacks) in some concrete cases [12].

Finally, the way secrets are handled is quite different. In spi this is achieved elegantly by means of the restriction operator, while in CryptoSPA we have explicitly to manage the set of pieces of information that are given to the enemies.

5.4 Comparing the Security Properties

One may wonder if the properties of secrecy and integrity defined in the two different process algebraic frameworks are somehow related. In spite of the technical differences, integrity is indeed the same property. The actual definition of integrity asks to check that the system meets its magical specification for each continuation $F(y)$, provided $F(y)$ does not reveal information on the message y. In [16], it has been shown that it is enough to consider the simple continuation $F(x) = \overline{received} x$ to establish whether a protocol enjoys integrity.

On the contrary, the two notions of secrecy are clearly different: the spi one is based on the idea of indistinguishablility, while the CryptoSPA one is based on the idea of possession, i.e. the so-called Dolev-Yao (see also [1]). To see the main point we must note that secret parameter x, in the process $S(x)$, must be a public value, i.e. a non-restricted one. Thus, among the possible tests we may found at least one that "knows" it, i.e. it has it as a free name. While, on the usual CryptoSPA approach the secret values are never "known" by the enemies; in fact, when an enemy is able to discover it we say that there is a secrecy attack.

Consider the process $P(x) = (\nu n)\overline{M}\langle n\rangle$. Note that x does not occur in the term of P. Thus, P necessarily preserves the secrecy of the public messages $M, M', \ldots$, indeed, $P(M) = P(M') = P$. However, from another point of view, P reveals one of its (declared) "secrets", i.e. the private name n, since it communicates n on the public channel M. In the usual CryptoSPA approach for secrecy, we would be interested in studying the secrecy of the restricted names, rather than the public ones.

The notion of secrecy developed in the spi approach is rather a form of information flow, i.e. non-interference. As a matter of fact, it can be formulated in the GNDC schema as follows:

– $S(x)$ preserves the secrecy of x iff $c(x).S(x)$ enjoys $NDC^{S(M)}_{\approx_{may}}$ (with M public) w.r.t. all the enemies whose sort is $\{c\}$ that output at least one message on the channel c and $c \notin Sort(S)$.

Thus, simply by considering other assumptions on the set of possible intruders it is possible to code secrecy in the spi as a GNDC property. Moreover, it is possible to show that if $S(x)$ preserves the secrecy of x and $S(M)$ enjoys $GNDC^{P}_{may}$ then also $S(M')$ enjoys $GNDC^{P}_{may}$. This holds because may testing may be defined in CryptoSPA and it is a congruence w.r.t. restriction and parallel composition (under certain assumptions, e.g. see [16]).

6 Other Frameworks

Very shortly we mention also other well-known process algebraic approaches that have been proposed in recent years.

The oldest and most widely deployed approach is the one based on CSP, which is well-illustrated in the book [33]. It shares similarities with CryptoSPA, as also in this approach security properties are modeled as observable events decorating the protocol, even if the idea of explicitly applying non-interference is not used. This approach has been mechanized, by using a compiler (called Casper [23] that translates protocol specifications into CSP code) and the FDR model checker; it has been enhanced to deal with symbolic reasoning (data independence) in [32]. Similarly, we have a compiler, called CVS [11] that translates specifications in a pre-dialect of CryptoSPA into SPA (i.e., CCS) code, and the Concurrency Workbench model checker. These analyses are approximated by considering an enemy that has limited memory and capability of generating new messages. More advanced symbolic semantics have been studied for CryptoSPA in [24]. In [25, 26], the usage of contexts (open systems) to describe the security analysis scenarios has been advocated. The language used is similar to CryptoSPA. However, differently from the GNDC approach, the correct specification is given through logical formulas and the treatment of the admissible enemies is done by reducing the verification problem to a validity one in the logic. Recently, the approach has been extended with symbolic techniques (see [27]). For a symbolic semantics of a spi-like language see, for instance, [6].

As spi is an extension of the π calculus with cryptographic primitive, similarly sjoin [2] extends the join calculus with constructs for encryption and decryption and with names that can be used as keys, nonces or other tags.

The applied pi calculus (see [18]) deals with the variety of different cryptosystems by adopting a general term algebra with an equality relation. This process calculus permits to describe cryptographic protocols using different cryptosystems. Thus, both applied pi and CryptoSPA recognize the necessity to manage uniformly different kind of cryptography that may be present in a complex protocol. The former exploits term algebras plus equality while the latter exploits a generic inference system.

A more recent approach is LySa [8], which is a very close relative of spi and pi-calculus. LySa mainly differs in two respects: (i) absence of channels (one global communication medium) and (*ii*) tests on values being received in communications as well as values being decrypted are directly embedded inside inputs and decryptions. A static analysis technology, based on Control Flow Analysis, has been applied to security protocols, expressed in LySa, providing a fully automatic and efficient tool. The same technology has been successfully used to analyse secrecy for pi [4] and spi [5].

Other process algebraic approaches not strictly related to the analysis of cryptographic protocols include the ambient calculus and the security pi calculus. The ambient calculus [7] is concerned with mobility of ambients (abstract collection of processes and objects that functions both as a unit of mobility and a unit of security). Communication takes place only inside an ambient, hence the hierarchy of nested ambients regulates who can communicate with who. Differently from spi, the security pi calculus [28] extends the π calculus with a new construct $[P]_\sigma$ denoting that process P is running at security level σ. This calculus is not very suited to talk about cryptographic protocols but is tailored for access control policies.

Acknowledgements. We would like to thank Nadia Busi and Marinella Petrocchi for helpful comments.

References

1. M. Abadi. Security protocols and specifications. In *Proc. Foundations of Software Science and Computation Structures*, volume 1578 of LNCS, pages 1–13, 1999.
2. M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. *Information and Computation*, 174(1):37–83, 2002.
3. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
4. C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Static analysis for the pi-calculus with applications to security. *Information and Computation*, 168:68–92, 2001.
5. C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Flow logic for dolev-yao secrecy in cryptographic processes. *Future Generation Computer Systems*, 18(6):747–756, 2002.
6. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Automata, Languages and Programming*, LNCS, pages 667–681, 2001.
7. L. Cardelli and A. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
8. C.Bodei, M. Buchholtz, P.Degano, F. Nielson, and H. R. Nielson. Automatic validation of protocol narration. In *Proceedings of The 16th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2003.
9. R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(1-2):83–133, 1984.

10. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(12):198–208, 1983.
11. A. Durante, R. Focardi, and R. Gorrieri. A compiler for analysing cryptographic protocols using non-interference. *ACM Transactions on Software Engineering and Methodology*, 9(4):489–530, 2000.
12. A. Durante, R. Focardi, and R. Gorrieri. Cvs at work: A report on new failures upon some cryptographic protocols. In *Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security*, LNCS 2052, 2001.
13. R. Focardi and R. Gorrieri. A classification of security properties. *Journal of Computer Security*, 3(1):5–33, 1995.
14. R. Focardi and R. Gorrieri. Classification of security properties (part i: Information flow). In *Foundations of Security Analysis and Design*, volume 2171 of LNCS, pages 331–396, 2001.
15. R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proceedings of 27th International Colloquium in Automata, Languages and Programming*, volume 1853 of LNCS, pages 354–372, 2000.
16. R. Focardi, R. Gorrieri, and F. Martinelli. A comparison of three authentication properties. *Theoretical Computer Science*, 291(3):285–327, 2003.
17. R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proceedings of World Congress on Formal Methods (FM'99)*, volume 1708 of LNCS, pages 794–813, 1999.
18. C. Fournet and M. Abadi. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.
19. A. Gordon and A. Jeffrey. Authenticity by typing in security protocols. In *Proceedings of The 14th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2001.
20. A. D. Gordon. Notes on nominal calculi for security and mobility. In *Foundations of Security Analysis and Design*, volume 2171 of LNCS, pages 262–330, 2001.
21. J. Halpern and R. van der Meyden. A logic for SDSI's linked local name spaces. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.
22. G. Lowe. Breaking and fixing the Needham Schroeder public-key protocol using FDR. In *Proceedings of Tools and Algorithms for the Construction and the Analisys of Systems*, volume 1055 of LNCS, pages 147–166. Springer Verlag, 1996.
23. G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.
24. F. Martinelli. Symbolic semantics and analysis for crypto-ccs with (almost) generic inference systems. In *Proceedings of the 27th international Symposium in Mathematical Foundations of Computer Sciences(MFCS'02)*, volume 2420 of LNCS, pages 519–531.
25. F. Martinelli. *Formal Methods for the Analysis of Open Systems with Applications to Security Properties.* PhD thesis, University of Siena, Dec. 1998.
26. F. Martinelli. Analysis of security protocols as *open* systems. *Theoretical Computer Science*, 290(1):1057–1106, 2003.

27. F. Martinelli. Symbolic partial model checking for security analysis. In *Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security*, LNCS, 2003. To appear.

28. J. Rieley and Matthew Hennessy. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Trans. on Progr. Lang. and Systems (TOPLAS)*, 24(5):566–591, 2002.

29. R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

30. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, 1992.

31. R. M. Needham and M. D. Schroder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

32. A. Roscoe and P. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7(2-3):147–190, 1999.

33. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.

34. P. Samarati and S. D. C. di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag, 2001.

35. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. CST–99–93, Department of Computer Science, University of Edinburgh, 1992. Also published as ECS–LFCS–93–266.

36. T. Woo and S. Lam. A semantic model for authentication protocols. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 178–194, 1993.