Lecture Notes in Computer Science 2946

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Riccardo Focardi Roberto Gorrieri (Eds.)

Foundations of Security Analysis and Design II

FOSAD 2001/2002 Tutorial Lectures

Springer

Volume Editors

Riccardo Focardi
Università Ca' Foscari di Venezia, Dipartimento di Informatica
Via Torino 155, 30172 Mestre (Venice), Italy
E-mail: focardi@dsi.unive.it

Roberto Gorrieri
Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura Anteo Zamboni 7, 40127 Bologna, Italy
E-mail: gorrieri@cs.unibo.it

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): D.4.6, C.2, K.6.5, K.4, D.3, F.3, E.3

ISSN 0302-9743
ISBN 3-540-20955-7 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 10985960 06/3142 5 4 3 2 1 0

International School on Foundations of Security Analysis and Design

17–29 September 2001, 23–27 September 2002, Bertinoro, Italy

Security is a fast-growing area of computer science, with increasing relevance to real-life applications such as Internet transactions and electronic commerce. Foundations for the analysis and the design of security aspects of these applications are badly needed in order to validate and prove (or guarantee) their correctness. Recently an IFIP Working Group on "Theoretical Foundations of Security Analysis and Design" was established (see http://www.dsi.unive.it/IFIPWG1_7/ for more details) in order to promote research and education in security-related issues.

One of the many initiatives of the IFIP WG 1.7 has been the creation of the "International School on Foundations of Security Analysis and Design" (FOSAD) that is held annually at the Residential Centre of the University of Bologna in Bertinoro, with the goal of disseminating knowledge in this critical area, especially for participants coming from less-favored and non-leading countries. The Residential Center (see http://www.centrocongressibertinoro.it/) is a former convent and episcopal fortress that has been transformed into a modern conference facility with computing services and Internet access.

The first edition of this school (FOSAD 2000) was very successful and the collection of tutorial lectures was published in Springer LNCS volume 2171. This second volume collects some of the tutorials given at the two successive schools (FOSAD 2001 and FOSAD 2002) that attracted many participants from all over the world.

This volume collects six tutorial lectures given at these two schools. More precisely:

- Alessandro Aldini, Mario Bravetti, Alessandra Di Pierro, Roberto Gorrieri, Chris Hankin and Herbert Wiklicky (Two Formal Approaches for Approximating Noninterference Properties);
- Carlo Blundo and Paolo D'Arco (The Key Establishment Problem);
- Michele Bugliesi, Giuseppe Castagna, Silvia Crafa, Riccardo Focardi, Vladimiro Sassone (A Survey of Name-Passing Calculi and Crypto-primitives);
- Roberto Gorrieri, Riccardo Focardi and Fabio Martinelli (Classification of Security Properties – Part II: Network Security);
- Rosario Gennaro (Cryptographic Algorithms for Multimedia Traffic);
- Hanne Riis Nielson, Flemming Nielson and Mikael Buchholtz (Security for Mobility).

We want to thank all the institutions that have supported the initiatives: CNR-IAT, ONR, Università Ca' Foscari di Venezia, Università di Bologna, Progetto MURST "Metodi Formali per la Sicurezza e il Tempo" (MEFISTO), and

EU-FET project MyThS: Models and Types for Security in Mobile Distributed Systems. Moreover, the school was held under the auspices of the European Association for Theoretical Computer Science (EATCS – Italian Chapter), the International Federation for Information Processing (IFIP – WG 1.7), and the European Educational Forum. Finally, we want to warmly thank the local organizers of the school, especially Alessandro Aldini, Andrea Bandini, Chiara Braghin and Elena Della Godenza.

November 2003

Riccardo Focardi
Roberto Gorrieri

Table of Contents

Two Formal Approaches for Approximating Noninterference Properties ... 1
Alessandro Aldini, Mario Bravetti, Alessandra Di Pierro, Roberto Gorrieri, Chris Hankin, and Herbert Wiklicky

The Key Establishment Problem ... 44
Carlo Blundo and Paolo D'Arco

A Survey of Name-Passing Calculi and Crypto-Primitives ... 91
Michele Bugliesi, Giuseppe Castagna, Silvia Crafa, Riccardo Focardi, and Vladimiro Sassone

Classification of Security Properties (Part II: Network Security) ... 139
Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli

Cryptographic Algorithms for Multimedia Traffic ... 186
Rosario Gennaro

Security for Mobility ... 207
Hanne Riis Nielson, Flemming Nielson, and Mikael Buchholtz

Author Index ... 267

Two Formal Approaches for Approximating Noninterference Properties

Alessandro Aldini[1], Mario Bravetti[2], Alessandra Di Pierro[3], Roberto Gorrieri[2], Chris Hankin[4], and Herbert Wiklicky[4]

[1] Istituto STI, Università di Urbino *Carlo Bo*, Italy
[2] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy
[3] Dipartimento di Informatica, Università di Pisa, Italy
[4] Department of Computing, Imperial College, London, UK

Abstract. The formalisation of security properties for computer systems raises the problem of overcoming also in a formal setting the classical view according to which confidentiality is an absolute property stating the complete absence of any unauthorised disclosure of information. In this paper, we present two formal models in which the notion of noninterference, which is at the basis of a large variety of security properties defined in the recent literature, is approximated. To this aim, the definition of indistinguishability of process behaviour is replaced by a similarity notion, which introduces a quantitative measure ε of the behavioural difference among processes. The first model relies on a programming paradigm called Probabilistic Concurrent Constraint Programming, while the second one is presented in the setting of a probabilistic process algebra. In both models, appropriate notions of distance provide information (the ε) on the security level of the system at hand, in terms of the capability of an external observer of identifying illegal interferences.

1 Introduction

The exact estimation of properties of computer systems is a problem that was widely and successfully attacked via several different formal approaches (see, e.g., [CT02,BHK01,HHHMR94,HS95,Hil96,BDG98,Ber99,BB00,Bra02]). However, a number of factors make the use of approximation techniques necessary to enhance the reliability of "exact" solutions obtained through the formal analysis of the mathematical model of a real, complex system. On the one hand, the confidence we can have in the answers computed by a software tool, which are delivered with certainty, strictly depends on the likelihood of obtaining precise information needed to formally specify the system at hand. On the other hand, even when such information is exact, the results of the mathematical analysis definitely assert that the considered property is or is not satisfied by the system model, while in practice it often happens that a system that approximately behaves like a perfect one is not only acceptable but also the only possible implementation. In practice, in a realistic scenario, a qualitative binary answer to the classical question "does the system satisfy my property?" is too restrictive and, in many cases, not significant.

R. Focardi and R. Gorrieri (Eds.): FOSAD 2001/2002, LNCS 2946, pp. 1–43, 2004.

In this work, we concentrate on formal techniques that employ probabilistic information to give a *quantitative* answer to the kind of question above in the restricted framework of security properties. Indeed, the motivations surveyed above apply also to the problem of verifying the security requirements of real systems. It is well-accepted that the unauthorised disclosure of confidential information cannot be completely avoided in real, complex systems, where typically the interplay between the portion of the system handling secrets and the other components that instead manage public information is more tight than that we expect [RMMG01]. In practice, part of the information flowing through the system cannot be controlled, and a portion of such an unavoidable information flow is sometimes illegal, in the sense that it reveals confidential data to unauthorised users. In such a case, the goal of the designer consists of minimising the illegal information leakage, and, as a consequence, the aim of the analyst must be the provision of an approximated estimation of such an information leakage. As a simple, real example, consider a password-based authentication system, like, e.g., an automatic teller machine. It is trivial to verify that absolute secrecy cannot be guaranteed. In fact, a brute-force based attack has the possibility, even if negligible, of guessing the password, thus violating the secrecy requirements. The analysis of such a kind of system is beyond the scope of possibilistic information flow techniques, which reject programs that do not guarantee absolute secrecy. A more interesting analysis should state that a potential information leakage is not troubling. From a quantitative viewpoint, this corresponds to verify whether or not the probability of detecting a potential illegal information flow is beyond a threshold for which the observer considers the system to be secure "enough". In case of the automatic teller machine, the probability of cracking the system depends on the length of the password and on the number of attempts at disposal of the attacker. By playing on these parameters, the designer can limit to a negligible (as desired) value the probability of accessing the system without knowing the appropriate password.

The approach to information flow analysis we consider is based on the idea of noninterference, originally proposed in [GM82], which states that "one group of users, using a certain set of commands, is noninterfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see". In a security context, the first group is represented by the high-level users, which execute confidential, secret activities, while the second group is given by the low-level users, which instead see public data only. The intuition is that the low-level view of the system to be analysed is not to be altered by the behaviour of the high-level users. If this is the case, we say that any covert channel cannot be set up from the high level to the low level. The verification of the condition above is based on the idea of indistinguishability of behaviours: in order to establish that there is no information flow between a high-level component H and a low-level object L, it is sufficient to check if for any pair of behaviours of the system that differ only in H's behaviour, L's observations cannot distinguish these two behaviours. Depending on the nature of the information flow, an external observer can characterise differ-

ent kinds of interference, due, e.g., to the deterministic, nondeterministic, timed, or probabilistic behaviour of the system. In particular, possibilistic noninterference for nondeterministic programs is weaker than probabilistic noninterference, which helps to reveal those covert channels that arise from the analysis of the frequency of the possible observations in several consecutive executions of the system [Gra90,McL90]. Consider, e.g., a program P that handles pin numbers needed to access the automatic teller machine mentioned above. At a certain point of the execution, the following statement is executed:

$$low\_variable := \mathrm{PIN}_i +^p \mathtt{rand}(9999)$$

where $+^p$ is a probabilistic choice operator that selects the left-hand command (which assigns a secret pin to a public, low variable) with probability p and the right-hand command (which assigns a random value from the range $[0 \ldots 9999]$ to the low variable) with probability $1 - p$. According to a purely nondeterministic behaviour, the program above is secure, since the set of possible outcomes does not change depending on which command will be executed. However, statistical inferences derived from the relative frequency of outcomes of repeated executions of the program allow an external observer to disclose the secret pin with a confidence that depends on the number of executed experiments.

Probabilistic noninterference also offers the means for approximating noninterference properties, by quantifying the real effectiveness of each possibilistic covert channel. More precisely, the key idea of an approach based on probabilistic noninterference is to replace the notion of indistinguishability by an appropriate notion of similarity. For instance, consider again program P and assume that parameter p is a value very close to 0. Obviously, the behaviour of P is not the same as that of the following secure program P':

$$low\_variable := \mathtt{rand}(9999)$$

since if we execute "infinitely often" both programs, then the limit of the frequencies of the possible outcomes allow the observer to distinguish P from P'. However, in practice we have that P and P' are similar and the probability of distinguishing the two programs is still negligible even after a large number n of experiments. In other words, P is considered to be an acceptable approximation of a secure program. As a result of an approach that replaces the restrictive idea of indistinguishability by a relaxed, more realistic notion of similarity, we can accept as secure systems a number of programs that somehow suffer from an information leakage but in practice offer sufficient security guarantees.

In this work, we survey two semantics-based security models (i.e., models that analyse the program behaviour to verify security properties) in which the notion of noninterference is approximated in the sense that they allow for some exactly quantified information leakage. The first one formalises such an approach in the context of a particular probabilistic declarative language, while the second one is based on a probabilistic process algebraic framework.

Language-based formalisms provide a suitable framework for analysing the confidentiality properties of real, complex computing systems. Particularly promising is the use of program semantics and analysis for the specification of

information-flow policies and information-flow controls which guarantee data confidentiality (see, e.g., [SM03] for a survey).

On the other hand, process algebras provide all the main ingredients needed to specify and analyse noninterference properties of computer systems (see, e.g., the several process algebraic approaches described in [FG01]). They are designed with the aim of describing concurrent systems that may interact through the exchange of messages, so that they can be used to naturally express each information flow occurring within the system to be modeled. They deal with both nondeterminism and, as we will focus in this work, probability, so that several kinds of information leakage can be revealed. They also deal in an elegant way with abstraction thanks to the hiding operator, which can be used to specify the observational power of each external observer, depending on the security level of such an observer. Last but not least, there exists a strong, natural similarity between the notion of indistinguishability for processes and semantic equivalences over process algebraic terms.

In the following (Sect. 2), we first introduce the language-based approach by presenting a formalisation of a noninterference property called *confinement* together with its probabilistic and approximated versions in the setting of the probabilistic programming language PCCP (Probabilistic Concurrent Constraint Programming) [DW98a,DW98b]. In this language nondeterminism is completely replaced by probabilistic choice, which makes it possible to develop a statistical interpretation of the approximation of the security property. Moreover, the different role played by variables in imperative and constraint programming hinders a direct translation of previous formalisation of noninterference based on the imperative paradigm into the PCCP setting, where a more appropriate notion must consider process identity rather than variables values.

Then (Sect. 3), we introduce a process algebraic framework for approximating probabilistic noninterference [ABG03]. The basic calculus integrates the characteristics of the classical CCS [Mil89] and CSP [Hoa85] and employs the probabilistic model introduced in [BA03], which is a mixture of the reactive and generative models of probability [GSS95]. Such an approach permits the modeler to specify both nondeterministic behaviour and probabilistic information in the same system model. The behavioural equivalence of process expressions is defined in terms of weak probabilistic bisimulation [BH97], a probabilistic extension of the classical weak bisimulation by Milner [Mil89]. Moreover, the behavioural similarity among processes is defined in terms of a relation called weak probabilistic bisimulation with ε-precision, an approximated version of the weak probabilistic bisimulation, where ε provides information on "how much" two behaviours differ from each other.

Finally (Sect. 4), some conclusions and comments about related work terminate the paper.

2 Language-Based Approach to Noninterference

2.1 Probabilistic Concurrent Constraint Programming

Probabilistic Concurrent Constraint Programming (PCCP) [DW98a,DW98b] is a probabilistic version of the Concurrent Constraint Programming (CCP) paradigm [SRP91,SR90]. This can be seen as a kind of process algebra enhanced with a notion of computational state. More precisely, CCP as well as PCCP are based on the notion of a generic *constraint system* $\mathcal{C}$, defined as a cylindric algebraic complete partial order (see [SRP91,dDP95] for more details), which encodes the information ordering. This is referred to as the *entailment* relation $\vdash$ and is sometimes denoted by $\sqsupseteq$. A cylindric constraint system includes constraints of the form $\exists_x c$ (cylindric elements) to model *hiding* of local variables, and constraints of the form d_{xy} (diagonal elements) to model *parameter passing*. The axioms of the constraint system include laws from the theory of cylindric algebras [HMT71] which model the cylindrification operators $\exists_x$ as a kind of first-order existential quantifiers, and the diagonal elements d_{xy} as the equality between x and y.

Table 1. The Syntax of PCCP Agents

$A ::= \mathbf{tell}(c)$	adding a constraint
$\mid \ []_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow p_i : A_i$	probabilistic choice
$\mid \ \|\|_{i=1}^{n} q_i : A_i$	prioritised parallelism
$\mid \ \exists_x A$	hiding, local variables
$\mid \ p(x)$	procedure call, recursion

In PCCP probability is introduced via a probabilistic choice and a form of probabilistic parallelism. The former replaces the nondeterministic choice of CCP, while the latter replaces the pure nondeterminism in the interleaving semantics of CCP by introducing a probabilistic scheduling. This allows us to implement mechanisms for differentiating the relative advancing speed of a set of agents running in parallel.

The concrete syntax of a PCCP agent A is given in Table 1, where c and c_i are *finite* constraints in $\mathcal{C}$, and p_i and q_i are real numbers representing probabilities. Note that at the syntactic level no restrictions are needed on the values of the numbers p_i and q_i; as explained in the next section, they will be turned into probability distributions by a normalisation process occurring during the computation. The meaning of $p(x)$ is given by a procedure declaration of the form $p(y) :- A$, where y is the formal parameter. We will assume that for each procedure name there is at most one definition in a fixed set of declarations (or program) P.

2.2 Operational Semantics

The operational model of PCCP can be intuitively described as follows. All processes share a common store consisting of the least upper bound, denoted by $\sqcup$, (with respect to the inverse $\sqsubseteq$ of the entailment relation) of all the constraints established up to that moment by means of **tell** actions. These actions allow for communication. Synchronisation is achieved via an **ask** guard which tests whether the store entails a given constraint. The probabilistic choice construct allows for a random selection of one of the different possible synchronisations making the program similar to a *random walk*-like stochastic process. Parts of the store can be made local by means of a *hiding operator* corresponding to a logical existential quantifier.

The operational semantics of PCCP is formally defined in terms of a probabilistic transition system, $(\mathrm{Conf}, \longrightarrow_p)$, where Conf is the set of configurations $\langle A, d\rangle$ representing the state of the system at a certain moment and the transition relation $\longrightarrow_p$ is defined in Table 2. The state of the system is described by the agent A which has still to be executed, and the common store d. The index p in the transition relation indicates the probability of the transition to take place. In order to describe all possible stages of the evolution of agents, in Table 2 we use an extended syntax by introducing an agent **stop** which represents successful termination, and an agent $\exists_x^d A$ which represents the evolution of an agent of the form $\exists_x B$ where d is the local information on x produced during this evolution. The agent $\exists_x B$ can then be seen as the particular case where the local store is empty, that is $d = \mathit{true}$. In the following we will identify all agents of the form $\|_{i=1}^n\, q_i : \mathbf{stop}$ and $\exists_x^d \mathbf{stop}$ with the agent **stop** as they all indicate a successful termination.

The rules of Table 2 are closely related to the ones for nondeterministic CCP, and we refer to [dDP95] for a detailed description. The rules for probabilistic choice and prioritised parallelism involve a normalisation process needed to redistribute the probabilities among those agents A_i which can actually be chosen for execution. Such agents must be enabled (i.e. the corresponding guards $\mathbf{ask}(c_i)$ succeed) or active (i.e. able to make a transition). This means that we have to re-define the probability distribution so that only enabled/active agents have non-zero probabilities and the sum of these probabilities is one. The probability after normalisation is denoted by $\tilde{p}_j$. For example, in rule **R2** the normalised transition probability can be defined for all enabled agents by

$$\tilde{p}_i = \frac{p_i}{\sum_{\vdash c_j} p_j},$$

where the sum $\sum_{\vdash c_j} p_j$ is over all enabled agents. When there are no enabled agents normalisation is not necessary. We treat a zero probability in the same way as a non-entailed guard, i.e. agents with zero probability are not enabled; this guarantees that normalisation never involves a division by a zero value. Analogous considerations apply to the normalisation of active agents in **R3**. It might be interesting to note that there are alternative ways to deal with the situation where $\sum_{\vdash c_j} p_j = 0$ (all enabled agents have probability zero). In

[DW00] normalisation is defined in this case as the assignment of a uniform distribution on the enabled agents; such a normalisation procedure allows, for example, to introduce a quasi-sequential composition.

The meaning of rule **R4** is intuitively explained by saying that the agent $\exists_x^d A$ behaves "almost" like A, with the difference that the variable x which is possibly present in A must be considered local, and that the information present in d has to be taken into account. Thus, if the store which is visible at the external level is c, then the store which is visible internally by A is $d \sqcup (\exists_x c)$. Now, if A is able to make a step, thus reducing itself to A' and transforming the local store into d', what we see from the external point of view is that the agent is transformed into $\exists_x^{d'} A'$, and that the information $\exists_x d$ present in the global store is transformed into $\exists_x d'$.

The semantics of a procedure call $p(x)$, modelled by Rule **R5**, consists in the execution of the agent A defining $p(x)$ with a parameter passing mechanism similar to call-by-reference: the formal parameter x is linked to the actual parameter y in such a way that y inherits the constraints established on x and vice-versa. This is realised in a way to avoid clashes between the formal parameter and occurrences of y in the agent via the operator Δ_y^x defined by:

$$\Delta_y^x A = \begin{cases} \exists_y^{d_{xy}} A \text{ if } x \neq y \\ A \qquad \text{if } x = y. \end{cases}$$

Table 2. The Transition System for PCCP

R1 $\langle \mathbf{tell}(c), d\rangle \longrightarrow_1 \langle \mathbf{stop}, c \sqcup d\rangle$

R2 $\langle []_{i=1}^n \mathbf{ask}(c_i) \to p_i : A_i, d\rangle \longrightarrow_{\tilde{p}_j} \langle A_j, d\rangle \quad j \in [1,n]$ and $d \vdash c_j$

R3 $$\frac{\langle A_j, c\rangle \longrightarrow_p \langle A_j', c'\rangle}{\langle \|_{i=1}^n \ p_i : A_i, c\rangle \longrightarrow_{p\cdot\tilde{p}_j} \langle \|_{j\neq i=1}^n p_i : A_i \parallel p_j : A_j', c'\rangle} \ j \in [1,n]$$

R4 $$\frac{\langle A, d \sqcup \exists_x c\rangle \longrightarrow_p \langle A', d'\rangle}{\left\langle \exists_x^d A, c\right\rangle \longrightarrow_p \left\langle \exists_x^{d'} A', c \sqcup \exists_x d'\right\rangle}$$

R5 $\langle p(y), c\rangle \longrightarrow_1 \langle \Delta_y^x A, c\rangle \qquad p(x) : -A \in P$

Observables. We will consider a notion of observables which captures the probabilistic input/output behaviour of a PCCP agent. We will define the observables $\mathcal{O}(A, d)$ of an agent A in store d as a probability distribution on constraints. Formally, this is defined as an element in the real vector space:

$$\mathcal{V}(\mathcal{C}) = \left\{ \sum x_c c \;\middle|\; x_c \in \mathbb{R},\ c \in \mathcal{C} \right\},$$

that is the free vector space obtained as the set of all formal linear combinations of elements in $\mathcal{C}$. The coefficients x_c represent the probability associated to constraints c.

Operationally, a distribution $\mathcal{O}(A, d)$ corresponds to the set of all pairs $\langle c, p \rangle$, where c is the result of a computation of A starting in store d and p is the probability of computing that result. For the purpose of this paper we will restrict to agents which only exhibit computations whose length is bounded. Note that since our programs are finitely branching this implies by König's lemma that the vector space of constraints is finite-dimensional.

We formally define the set of results for an agent A as follows.

Definition 1. *Let A be a PCCP agent. A computational path π for A in store d is defined by*

$$\pi \equiv \langle A_0, c_0 \rangle \longrightarrow_{p_1} \langle A_1, c_1 \rangle \longrightarrow_{p_2} \ldots \longrightarrow_{p_n} \langle A_n, c_n \rangle ,$$

where $A_0 = A$, $c_0 = d$, $A_n = \mathbf{stop}$ and $n < \infty$.

We denote by $\mathrm{Comp}(A, d)$ the set of all computational paths for A in store d.

Definition 2. *Let $\pi \in \mathrm{Comp}(A, d)$ be a computational path for A in store d,*

$$\pi \equiv \langle A, d \rangle = \langle A_0, c_0 \rangle \longrightarrow_{p_1} \langle A_1, c_1 \rangle \longrightarrow_{p_2} \ldots \longrightarrow_{p_n} \langle A_n, c_n \rangle .$$

We define the result of π as $res(\pi) = c_n$ and its probability as $prob(\pi) = \prod_{i=1}^{n} p_i$.

Because of the probabilistic choice, there might be different computational paths for a given PCCP program which lead to the same result. The probability associated to a given result c is then the sum of all probabilities $prob(\pi)$ associated to all paths π such that $res(\pi) = c$. This suggests that we introduce the following equivalence relation on $\mathrm{Comp}(A)$.

Definition 3. *Let $\pi, \pi' \in \mathrm{Comp}(A)$ be two computational paths for A in store d. We define $\pi \approx \pi'$ iff $res(\pi) = res(\pi')$. The equivalence class of π is denoted by $[\pi]$.*

The definitions of $res(\pi)$ and $prob(\pi)$ are extended to $\mathrm{Comp}(A)_{/\approx}$ in the obvious way by $res([\pi]) = res(\pi)$ and $prob([\pi]) = \sum_{\pi' \in [\pi]} prob(\pi')$.

We can now define the probabilistic input/output observables of a given agent A in store d as the set

$$\mathcal{O}(A, d) = \left\{ \langle res([\pi]), prob([\pi]) \rangle \,|\, [\pi] \in \mathrm{Comp}(A)_{/\approx} \right\} .$$

In the following we will adopt the convention that whenever the initial store is omitted then it is intended to be *true*.

Example 1. [CHM02] Consider an ATM (Automatic Teller Machine) accepting only a single PIN number n out of m possible PINs, e.g. $m = 10000$:

$$\begin{aligned} \mathrm{ATM}n \equiv & \Big[\Big]_{i=1, i \neq n}^{m} \mathbf{ask}(\mathrm{PIN}i) \to 1 : \mathbf{tell}(alarm) \\ & [\!] \ \mathbf{ask}(\mathrm{PIN}n) \to 1 : \mathbf{tell}(cash) \end{aligned}$$

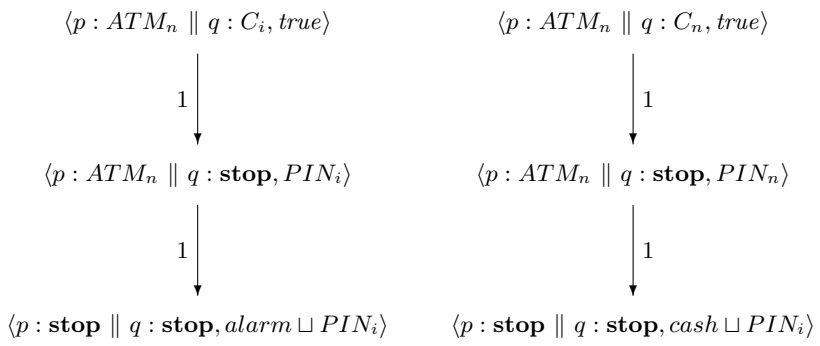


Fig. 1. Execution of a program simulating the interaction with an ATM

This agent simulates an ATM which recognises PINn: if PINn has been told the machine dispenses *cash*, otherwise — for any incorrect PINi — it sounds an *alarm*.

Consider now the following agent representing the client whose PIN is i:

$$C_i \equiv \mathbf{ask}(true) \rightarrow 1 : \mathbf{tell}(\mathrm{PIN}i).$$

The computational paths for the parallel composition $M_i \equiv p : \mathrm{ATM}n \parallel q : C_i$ are given in Figure 1 respectively for the case in which $i = n$ and $i \neq n$. When run in the initial store *true*, agent ATMn is not active (no constraints PINj, $1 \leq j \leq m$ is entailed by the store); thus C_i is scheduled with probability 1 (obtained by q after normalisation). The resulting configuration is $\langle p : \mathrm{ATM}n \parallel q : \mathbf{stop}, \mathrm{PIN}i\rangle$. Now the only active agent is ATMn which is then executed with (normalised) probability 1 leading to the final configuration $\langle \mathbf{stop}, cash\rangle$ in the case where the PIN number is correct (right hand side derivation in Figure 1) and $\langle \mathbf{stop}, alarm\rangle$ in the case where the PIN number is wrong (left hand side derivation in Figure 1).

The observables are then $\mathcal{O}(M_i) = \{\langle \mathrm{PIN}_i \sqcup alarm, 1\rangle\}$ for $i \neq n$, $\mathcal{O}(M_i) = \{\langle \mathrm{PIN}_n \sqcup cash, 1\rangle\}$ for $i = n$.

2.3 Probabilistic Noninterference and Identity Confinement

The original idea of noninterference as stated in [GM82] can be expressed in the PCCP formalism via the notion of *identity confinement*. Roughly, this notion establishes whether it is possible to identify which process is running in a given program. Therefore, given a set of agents and a set of potential intruders, the latter cannot see what the former set is doing, or more precisely, no spy is able to find out which of the agents in the first group is actually being executed. This formulation is the natural translation in the context of PCCP of the notion of confinement typically expressed in imperative languages via the values of variables [SS00].

The following example illustrates the notion of identity confinement as compared to the imperative formulation. It also shows the difference between nondeterministic and probabilistic (identity) confinement.

Example 2. In an imperative language, confinement — as formulated for example in [SS99,SS00] — usually refers to a standard two-level security model consisting of high and low level variables. One then considers the (value of the) high variable h as confined if the value of the low level variable l is not "influenced" by the value of the high variable, i.e. if the observed values of l are independent of h.

The following statement illustrates the difference between nondeterministic and probabilistic confinement:

$$h := h \bmod 2;\ \ (l := h \ {}_{\frac{1}{2}}[]_{\frac{1}{2}}\ (l := 0 \ {}_{\frac{1}{2}}[]_{\frac{1}{2}}\ l := 1))$$

The value of l clearly depends "somehow" on h. However, if we resolve the choice nondeterministically it is impossible to say anything about the value of h by observing the possible values of l. Concretely, we get the following dependencies between h and possible values of l:

- For $h \bmod 2 = 0$: $\{l = 0, l = 1\}$
- For $h \bmod 2 = 1$: $\{l = 1, l = 0\}$,

i.e. the possible values of l are the same independently from the fact that h is even or odd. In other words, h is nondeterministically confined.

In a probabilistic setting the observed values for l and their probabilities allow us to distinguish cases where h is even from those where h is odd. We have the following situation:

- For $h \bmod 2 = 0$: $\left\{\left\langle l = 0, \frac{3}{4}\right\rangle, \left\langle l = 1, \frac{1}{4}\right\rangle\right\}$
- For $h \bmod 2 = 1$: $\left\{\left\langle l = 0, \frac{1}{4}\right\rangle, \left\langle l = 1, \frac{3}{4}\right\rangle\right\}$

Therefore, the probabilities to get $l = 0$ and $l = 1$ reveal if h is even or odd, i.e. h is probabilistically *not* confined.

Example 3. We can re-formulate the situation above in our declarative setting by considering the following agents:

$$\texttt{hOn} \equiv \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \mathbf{tell}(\texttt{on})\ []\ \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \texttt{Rand}$$
$$\texttt{hOff} \equiv \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \mathbf{tell}(\texttt{off})\ []\ \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \texttt{Rand}$$
$$\texttt{Rand} \equiv \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \mathbf{tell}(\texttt{on})\ []\ \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \mathbf{tell}(\texttt{off})$$

The constraint system consists of four elements:

$$\mathcal{C} = \{\mathit{true}, \texttt{on}, \texttt{off}, \mathit{false} = \texttt{on} \sqcup \texttt{off}\},$$

where $\mathit{true} \sqsubseteq \texttt{on} \sqsubseteq \mathit{false}$ and $\mathit{true} \sqsubseteq \texttt{off} \sqsubseteq \mathit{false}$.

The constraints `on` and `off` represent the situations in which the low variable $l = 1$ or $l = 0$ respectively. The agent `hOn` corresponds then to the behaviour

of the imperative program fragment in case that $h \mod 2 = 1$, while `hOff` corresponds to the case where $h \mod 2 = 0$. The auxiliary agent `Rand` corresponds to the second choice in the above imperative fragment. The imperative notion of confinement now translate in our framework into a problem of identity confinement: getting information about h in the previous setting is equivalent to discriminating between `hOn` and `hOff`, i.e. revealing their identity. The two agents will be identity confined if they are observationally equivalent in any context.

As explained in Section 2.2, the observables of a PCCP agent correspond to a distribution on the constraint system, that is a vector in the space $\mathcal{V}(\mathcal{C})$. Thus, the difference between two observables corresponds to the vector difference between the given observables and can be measured by means of a *norm*. We adopt here the supremum norm $\|\cdot\|_\infty$ formally defined as

$$\|(x_i)_{i\in I}\|_\infty = \sup_{i\in I} |x_i|,$$

where $(x_i)_{i\in I}$ represents a probability distribution. However, as long as we are interested in defining the identity of two vectors, any p-norm: $\|(x_i)_{i\in I}\|_p = \sqrt[p]{\sum_{i\in I} |x_i|^p}$ would be appropriate.

Probabilistic identity confinement is then defined as follows [DHW01]:

Definition 4. *Two agents A and B are probabilistically identity confined iff their observables are identical in any context, that is for all agent S,*

$$\mathcal{O}(p : A \parallel q : S) = \mathcal{O}(p : B \parallel q : S)$$

or equivalently,

$$\Big\|\mathcal{O}(p : A \parallel q : S) - \mathcal{O}(p : B \parallel q : S)\Big\| = 0,$$

for all scheduling probabilities p and $q = 1 - p$.

Example 4. It is easy to check that any context can distinguish between the agents `hOn` and `hOff` of Example 3. In fact, even if executed on their own their observables are different (cf. Figure 2):

$$\Big\|\mathcal{O}(\texttt{hOn}, \mathit{true}) - \mathcal{O}(\texttt{hOff}.\mathit{true})\Big\| =$$
$$\left\| \left\{ \left\langle \texttt{on}, \frac{3}{4} \right\rangle, \left\langle \texttt{off}, \frac{1}{4} \right\rangle \right\} - \left\{ \left\langle \texttt{on}, \frac{1}{4} \right\rangle, \left\langle \texttt{off}, \frac{3}{4} \right\rangle \right\} \right\| = \frac{1}{2}.$$

Therefore `hOn` and `hOff` are *not* probabilistically identity confined.

Example 5. Consider the following two PCCP agents [DHW01]:

$$A \equiv \frac{1}{2} : \mathbf{tell}(c) \parallel \frac{1}{2} : \mathbf{tell}(d)$$

$$B \equiv \mathbf{tell}(c \sqcup d).$$

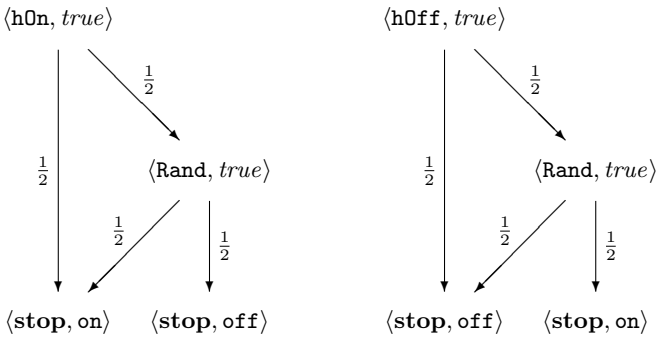


Fig. 2. Transitions for `hOn` and `hOff`

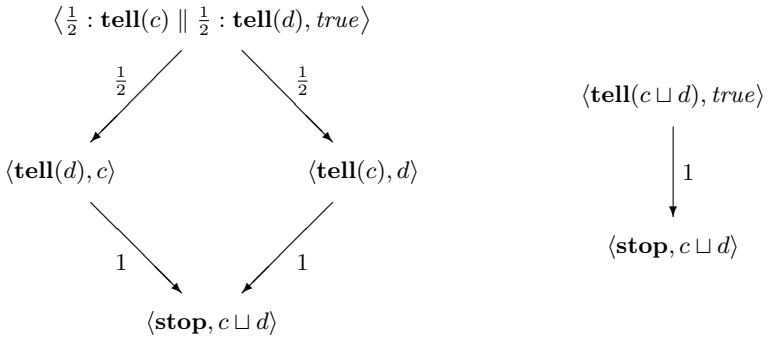


Fig. 3. Independent Executions of A and B

It is easy to see that in their nondeterministic versions A and B executed in any context give the same observables. A and B are thus nondeterministically identity confined.

Treating the choice probabilistically still gives us the same observables for A and B if they are executed on their own (cf. Figure 3), but they are not probabilistically confined. A context which reveals the identity of A and B is for example the agent:

$$C \equiv \mathbf{ask}(c) \to \frac{2}{3} : \mathbf{tell}(e) \,[]\, \mathbf{ask}(d) \to \frac{1}{3} : \mathbf{tell}(f),$$

as the executions of A and B in this context give different observables (cf. Figure 4 and Figure 5):

$$\mathcal{O}\left(\frac{1}{2} : A \parallel \frac{1}{2} : C\right) = \left\{\left\langle c \sqcup d \sqcup e, \frac{7}{12}\right\rangle, \left\langle c \sqcup d \sqcup f, \frac{5}{12}\right\rangle\right\}$$

$$\mathcal{O}\left(\frac{1}{2} : B \parallel \frac{1}{2} : C\right) = \left\{\left\langle c \sqcup d \sqcup e, \frac{2}{3}\right\rangle, \left\langle c \sqcup d \sqcup f, \frac{1}{3}\right\rangle\right\}.$$

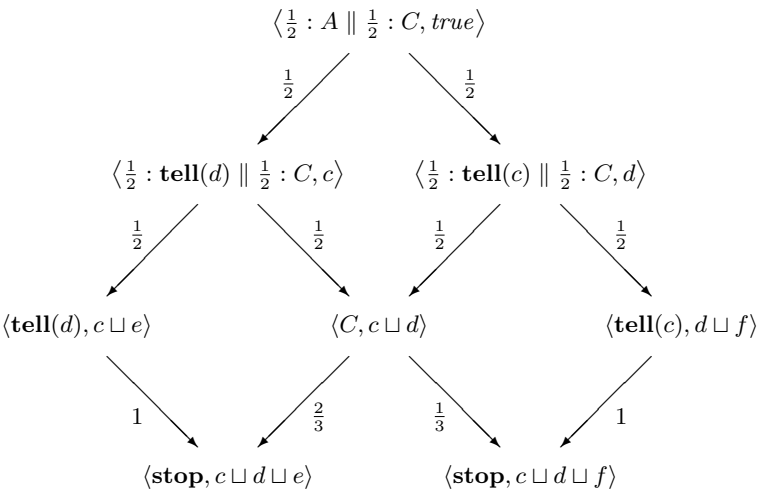


Fig. 4. Executions of A in Context C

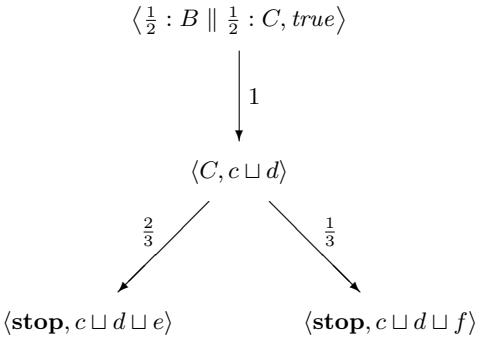


Fig. 5. Executions of B in Context C

We observe that if we restrict to a particular class of contexts, namely those of the form:

$$D \equiv \mathbf{ask}(g) \rightarrow 1 : \mathbf{tell}(h),$$

then A and B are probabilistically identity confined with respect to these agents: for any choice of the scheduling probabilities p and $q = 1-p$, we obtain the same observables for the parallel compositions of D with A and B respectively.

If neither c nor d entails g then D will never be executed, and the executions of $p : A \parallel q : D$ and $p : B \parallel q : D$ are essentially the same as for A and B alone (cf. Figure 3).

If only d entails g we obtain the derivations in Figure 6. The case where g is entailed by c alone is analogous. In all cases we end up with a single result $c \sqcup d \sqcup h$ with probability one.

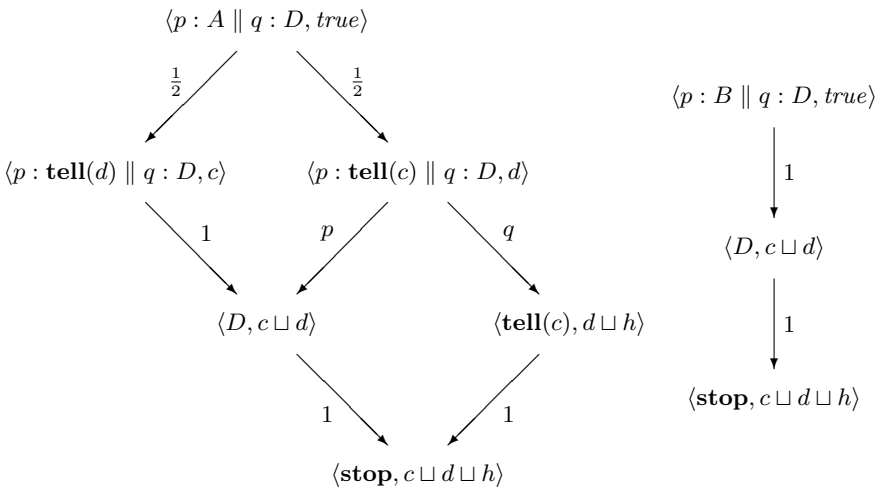


Fig. 6. Executions in Context D when d entails g

The derivations of $p : A \parallel q : D$ and $p : B \parallel q : D$ in the case that both c and d entail g are depicted in Figure 7: again we obtain the same result $c \sqcup d \sqcup h$ with probability one.

In general, identical behaviour in all contexts is hardly ever achievable. It therefore makes sense to ask for identical observables if A and B are executed in parallel with agents with only limited capabilities. Moreover, the power of a context can be evaluated in terms of its ability to distinguish the behaviours of two agents. It is also reasonable to think that its effectiveness will depend on the probabilities of the scheduling in the interleaving with the given agents. This leads to the definition of a weaker (and yet more practical) notion of probabilistic identity confinement which is parametric in the type of context S and the scheduling probability p. We will introduce such a notion, which we call *approximate identity confinement*, in the next section.

2.4 Approximate Identity Confinement

In Section 1 we argued that it is practically more useful to base noninterference properties on some *similarity* notions instead of equivalence once.

The confinement notion discussed above is *exact* in the sense that it refers to the equivalence of the agents' behaviour. In this section, we introduce a technique which allows us to relax confinement to an approximate and yet more effective notion.

The intuitive idea behind such a notion is that we look at *how much* the behaviours of two agents differ, instead of qualitatively asserting whether they are identical or not. In particular, in the probabilistic case we can measure the distance ε between the distributions representing the agents' observables instead of checking whether this difference is 0. We can then say that the agents are ε-confined for some $\varepsilon \geq 0$.

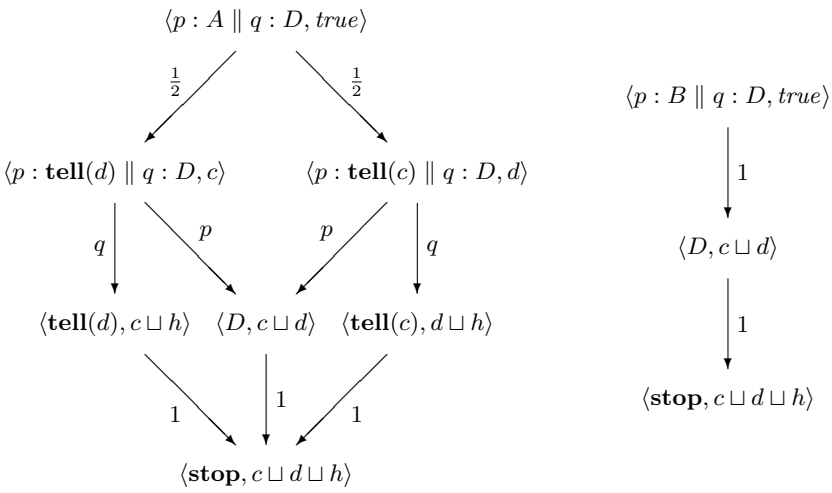


Fig. 7. Executions in Context D when both c and d entail g

We illustrate this idea by means of the ATM example introduced in Section 2.2.

Example 6. Consider the program in Example 1 which simulates an ATM (Automatic Teller Machine) accepting only a single PIN number n out of m possible PINs, e.g. $m = 10000$:

$$\begin{aligned}\mathrm{ATM}n \equiv &\Big[\Big]_{i=1,i\neq n}^{m} \mathbf{ask}(\mathrm{PIN}i) \to 1 : \mathbf{tell}(alarm) \\ &[\!] \ \mathbf{ask}(\mathrm{PIN}n) \to 1 : \mathbf{tell}(cash)\end{aligned}$$

The following agent simulates a spy which tries a random PIN number i:

$$S \equiv \Big[\Big]_{i=1}^{m} \mathbf{ask}(true) \to 1 : \mathbf{tell}(\mathrm{PIN}i)$$

If we consider two such machines $\mathrm{ATM}n_1$ and $\mathrm{ATM}n_2$ for $n_1 \neq n_2$ and execute them in context S we obtain two slightly different observables, namely:

$$\begin{aligned}\mathcal{O}\left(p : \mathrm{ATM}n_1 \parallel q : S\right) = &\left\{\left\langle \mathrm{PIN}n_1 \sqcup cash, \frac{1}{m}\right\rangle\right\} \\ &\cup \bigcup_{i=1,i\neq n_1}^{m} \left\{\left\langle \mathrm{PIN}i \sqcup alarm, \frac{1}{m}\right\rangle\right\}\end{aligned}$$

and

$$\begin{aligned}\mathcal{O}\left(p : \mathrm{ATM}n_2 \parallel q : S\right) = &\left\{\left\langle \mathrm{PIN}n_2 \sqcup cash, \frac{1}{m}\right\rangle\right\} \\ &\cup \bigcup_{i=1,i\neq n_2}^{m} \left\{\left\langle \mathrm{PIN}i \sqcup alarm, \frac{1}{m}\right\rangle\right\}.\end{aligned}$$

Clearly, $\mathcal{O}(p : \mathrm{ATM}n_1 \parallel q : S)$ and $\mathcal{O}(p : \mathrm{ATM}n_2 \parallel q : S)$ are different.

For most PINs both machines will sound an alarm in most cases, but if we are lucky, the spy will use the correct PINs in which case we are able to distinguish the two machines (besides earning some *cash*). The chances for this happening are small but are captured essentially if we look at the difference between the observables:

$$\left\| \mathcal{O}(p : \mathrm{ATM}n_1 \parallel q : S) - \mathcal{O}(p : \mathrm{ATM}n_2 \parallel q : S) \right\| = \frac{1}{m}.$$

The set $\{\mathrm{ATM}n\}_n$ is ε-confined with respect to S with $\varepsilon = \frac{1}{m}$ but not strictly confined. In the practical applications, m is usually very large, that is ε is very small, which makes it reasonable to assume the ATM's agents as secure although not exactly confined.

The notion of approximate identity confinement we will define in the following is based on the idea of measuring how much the behaviour of two agents differs if we put them in a certain context. We will refer to such a context as *spy* or *attacker*. This restriction makes sense as no system is secure against an omnipotent attacker [LMMS98] and its security depends on the quality of the possible attacker. We will discuss in the following different kinds of such attackers.

As an example, consider the class of attackers expressed in PCCP by:

$$\mathcal{S}_n = \left\{ \Box_{i=1}^{n} \mathbf{ask}(c_i) \to p_i : \mathbf{tell}(f_i) \right\},$$

where $f_i \in \mathcal{C}$ are *fresh* constraints, that is constraints which never appear in the execution of the host agents, and $c_i \in \mathcal{C}$. These agents are passive and memoryless attackers. They do not change the behaviour of the hosts, and are only allowed to interact with the store in one step. Nevertheless, they are sufficient for formalising quite powerful attacks such as the timing attacks in [Koc95].

A generalisation of this class is to consider active spies (e.g. Example 7 and Example 1) and/or spies with memory such as $\mathbf{ask}(c) \to p : \mathbf{ask}(d) \to q : \mathbf{tell}(f)$.

Example 7. Consider the two agents:

$$A \equiv \mathbf{ask}(c) \to 1 : \mathbf{tell}(d)$$
$$B \equiv \mathbf{stop}.$$

If executed in store *true*, A and B are obviously confined with respect to any *passive* spy. They both do nothing, and it is therefore impossible to distinguish them by just observing. However, for an *active* spy like $S \equiv \mathbf{tell}(c)$ it is easy to determine if it is being executed in parallel with A or B. Note that if executed in any store d such that $d \vdash c$, the two agents A and B are always distinguishable because their observables are different.

The notion of approximate confinement which we introduce in the following is a generalisation of the identity confinement introduced in [DHW01] and defined in Section 2.3. The definition we give is parametric with respect to a set of

admissible spies $\mathcal{S}$ and scheduling probabilities p and $q = 1-p$. We say that two agents A and B are approximately confined with respect to a set of spies $\mathcal{S}$ iff there exists an $\varepsilon \geq 0$ such that for all $S \in \mathcal{S}$ the *distance* between the observables of $p : A \parallel q : S$ and $p : B \parallel q : S$ is smaller than ε. We consider as a measure for this distance the supremum norm $\|\cdot\|_\infty$ as in Definition 4. In this case, the choice of this norm is particularly appropriate because it allows us to identify a single constraint c for which the associated probabilities are maximally different. In the following we will usually omit the index ∞.

Definition 5. *Given a set of admissible spies $\mathcal{S}$, we call two agents A and B ε-confined for some $\varepsilon \geq 0$ iff:*

$$\sup_{S \in \mathcal{S}} \left\| \mathcal{O}(p : A \parallel q : S) - \mathcal{O}(p : B \parallel q : S) \right\| = \varepsilon.$$

This definition can be generalised to a set of more than two agents.

The number ε associated to a given class of spies $\mathcal{S}$ can be seen as a measure of the "power" of $\mathcal{S}$. In fact, it is strongly related to the number of tests a spy needs to perform in order to reveal the identity of the host agents. We will make this argument more precise in the next section. Note that this number depends on the scheduling probability. This is because the effectiveness of a spy can only be evaluated depending on the internal design of the host system which is in general not known to the spy. For example, in [DHW03b] we have presented an analysis which shows that the "best" spy of the class $\mathcal{S}_2$ defined above is one with a choice distribution where p_1 is very close to 0 and p_2 is very close to 1, or vice versa.

Obviously, if two agents A and B are ε-confined with $\varepsilon(p) = 0$ for all scheduling probability p then they are probabilistically identity confined.

2.5 Statistical Interpretation

The notion of approximate confinement is strongly related to statistical concepts, in particular to so-called *hypothesis testing* (see e.g. [Sha99]).

Identification by Testing. Let us consider the following situation. We have two agents A and B which are attacked by a spy S. Furthermore, we assume that A and B are ε-confined with respect to S. This means that the observables $\mathcal{O}(p : A \parallel q : S)$ and $\mathcal{O}(p : B \parallel q : S)$ are ε-similar. In particular, as the observables do not include infinite results, we can identify some constraint $c \in \mathcal{C}$ such that $|p_A(c) - p_B(c)| = \varepsilon$, where $p_A(c)$ is the probability of the result c in an execution of $p : A \parallel q : S$ and $p_B(c)$ is the probability that c is a result of $p : B \parallel q : S$.

Following the standard interpretation of probabilities as "long-run" relative frequencies, we can thus expect that the number of times we get c as result of an execution of $p : A \parallel q : S$ and $p : B \parallel q : S$ will differ "on the long run" by exactly a factor ε. That means if we execute $p : A \parallel q : S$ or $p : B \parallel q : S$ "infinitely" often we can determine $p_A(c)$ and $p_B(c)$ as the limit of the frequencies with which we obtain c as result.

In fact, for any unknown agent X we can attempt do determine $p_X(c)$ experimentally by executing $p : X \parallel q : S$ over and over again. Assuming that X is actually the same as either A or B we know that the $p_X(c)$ we obtain must be either $p_A(c)$ or $p_B(c)$. We thus can easily determine this way if $X = A$ or $X = B$, i.e. reveal the identity of X (if $\varepsilon \neq 0$), simply by testing.

Unfortunately — as J.M. Keynes pointed out — we are all dead on the long run. The above described experimental setup is therefore only of theoretical value. For practical purposes we need a way to distinguish A and B by finite executions of $p : A \parallel q : S$ and $p : B \parallel q : S$. If we execute $p : A \parallel q : S$ and $p : B \parallel q : S$ only a finite number of — say n — times, we can observe a certain experimental frequency $p^n_A(c)$ and $p^n_B(c)$ for getting c as a result. Each time we repeat this finite sequence of n executions we may get different values for $p^n_A(c)$ and $p^n_B(c)$ (only the infinite experiments will eventually converge to the same constant values $p_A(c)$ and $p_B(c)$).

Analogously, we can determine the frequency $p^n_X(c)$ for an unknown agent X by testing, i.e. by looking at n executions of $p : X \parallel q : S$. We can then try to compare $p^n_X(c)$ with $p^n_A(c)$ and $p^n_B(c)$ or with $p_A(c)$ and $p_B(c)$ in order to find out if $X = A$ or $X = B$. Unfortunately, there is neither a single value for either $p^n_X(c)$, $p^n_A(c)$ or $p^n_B(c)$ (each experiment may give us different values) nor can we test if $p^n_X(c) = p^n_A(c)$ or $p^n_X(c) = p^n_B(c)$ nor if $p^n_X(c) = p_A(c)$ or $p^n_X(c) = p_B(c)$.

For example, it is possible that c is (coincidentally) not the result of the first execution of $p : X \parallel q : S$, although the (long-run) probabilities of obtaining c by executing $p : A \parallel q : S$ or $p : B \parallel q : S$ are, let's say, $p_A = 0.1$ and $p_B = 0.5$. If we stop our experiment after $n = 1$ executions we get $p^1_X(c) = 0$. We know that $X = A$ or $X = B$ but the observed $p^1_X(c)$ is different from both p_A and p_B.

Nevertheless, we could argue that it is more likely that $X = A$ as the observed $p^1_X(c) = 0$ is closer to $p_A = 0.1$ than to $p_B = 0.5$. The problem is now to determine, on the basis of such experiments, how much the identification of X with A is "more correct" than identifying X with B on the basis of such experiments.

For finite experiments we can only make a guess about the true identity of X, but never definitely reveal its identity. The *confidence* we can have in our guess or *hypothesis* about the identity of an unknown agent X — i.e. the probability that we make a correct guess — depends obviously on two factors: the number of tests n and the difference ε between the observables of $p : A \parallel q : S$ and $p : B \parallel q : S$.

Hypothesis Testing. The problem is to determine experimentally if the unknown agent X is one of two known agents A and B. The only way we can obtain information about X is by executing it in parallel with a spy S. In this way we can get an experimental estimate for the observables of $p : X \parallel q : S$. We then can compare this estimate with the observables of $p : A \parallel q : S$ and $p : B \parallel q : S$.

That means: based on the outcome of some finite experiments (involving an unknown agent X) we formulate a hypothesis H about the identity of X, namely either that "X is A" or that "X is B". Our hypothesis about the identity of X

will be formulated according to a simple rule: depending if the experimental estimate for the observables of $p : X \parallel q : S$ are closer to $\mathcal{O}(p : A \parallel q : S)$ or to $\mathcal{O}(p : B \parallel q : S)$ we will identify X with A or B respectively.

More precisely, the method to formulate the hypothesis H about the identity of the unknown process X consists of the two following steps:

1. We execute $p : X \parallel q : S$ exactly n times in order to obtain an experimental approximation, i.e. average, for its observables
$$\overline{\mathcal{O}}_n(p : X \parallel q : S) = \left\{ \left\langle c, \frac{\#\text{ of times } c \text{ is the result}}{n} \right\rangle \right\}_{c \in \mathcal{C}},$$
2. Depending if $\overline{\mathcal{O}}_n(p : X \parallel q : S)$ is closer to the observables $\mathcal{O}_n(p : A \parallel q : S)$ or $\mathcal{O}_n(p : B \parallel q : S)$ we formulate the hypothesis
$$H : \begin{cases} X = A \;\text{ if } \left\| \mathcal{O}_n(p : X \parallel q : S) - \mathcal{O}(p : A \parallel q : S) \right\| \\ \qquad\qquad \leq \left\| \mathcal{O}_n(p : X \parallel q : S) - \mathcal{O}(p : B \parallel q : S) \right\| \\ X = B \;\text{ otherwise.} \end{cases}$$

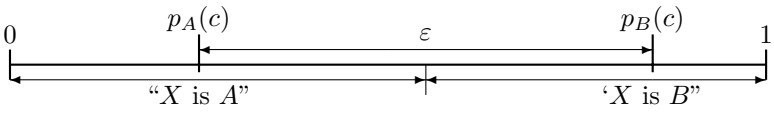
The question is now whether the guess expressed by the hypothesis H about the true identity of the black box X, which we formulate according to the above procedure, is correct; or more precisely: what is the probability that the hypothesis H holds? To do this we have to distinguish two cases or scenarios:

X **is actually** A**:** What is the probability (in this case) that we formulate the *correct* hypothesis $H : X$ is A and what is the probability that we formulate the *incorrect* hypothesis $H : X$ is B?

X **is actually** B**:** What is the probability (in this case) that we formulate the *correct* hypothesis $H : X$ is B and what is the probability that we formulate the *incorrect* hypothesis $H : X$ is A?

Clearly, in each case the probability to formulate a correct hypothesis and the probability to formulate an incorrect hypothesis add up to one. Furthermore, it is obvious that both scenarios "X is actually A" and "X is actually B" are symmetric. We will therefore investigate only one particular problem. Suppose that X is actually agent A, what is the probability that — according to the above procedure — we formulate the — in this case — correct hypothesis $H : X$ is A.

In the following we use the notation $p_X(c)$ and $p^n_X(c)$ to denote the probability assigned to $c \in \mathcal{C}$ in the distribution representing the observables $\mathcal{O}(p : X \parallel q : S)$ and in the experimental average $\mathcal{O}_n(p : X \parallel q : S)$ respectively. Furthermore, we look at a simplified situation where we are considering only a single constraint c where the difference between $p_A(c)$ and $p_B(c)$ is maximal. Let us assume without loss of generality that $p_A(c) < p_B(c)$ as in the diagram below:



If the experimental value $p^n_X(c) = p^n_A(c)$ we obtained in our test is anywhere to the left of $p_A(c) + \varepsilon/2$ then the hypothesis H we formulate (based on $p^n_A(c)$) will be the correct one: "X is A"; if the experimental value is to the right of $p_A(c) + \varepsilon/2$ we will formulate the incorrect hypothesis: "X is B".

Under the assumption that "X is actually A" the probability $\mathbf{P}(H)$ that we will formulate the correct hypothesis "X is A" is therefore:

$$\mathbf{P}\left(p^n_A(c) < p_A(c) + \frac{\varepsilon}{2}\right) = 1 - \mathbf{P}\left(p_A(c) + \frac{\varepsilon}{2} < p^n_A(c)\right).$$

To estimate $\mathbf{P}(H)$ we have just to estimate the probability $\mathbf{P}(p^n_A(c) < p_A(c) + \varepsilon/2)$, i.e. that the experimental value $p^n_A(c)$ will be left of $p_A(c) + \varepsilon/2$.

Confidence Estimation. The confidence we can have in the hypothesis H we formulate is true can be determined by various statistical methods. These methods allow us to estimate the probability that an experimental average X_n — in our case $p^n_A(c)$ — is within a certain distance from the corresponding expectation value $\mathbf{E}(X)$ — here $p_A(c)$ — i.e. the probability

$$\mathbf{P}\left(|X_n - \mathbf{E}(X)| \leq \varepsilon\right)$$

for some $\varepsilon \geq 0$. These statistical methods are essentially all based on the *central limit theorem*, e.g. [Bil86,GS97,Sha99].

The type of tests we consider here to formulate a hypothesis about the identity of the unknown agent X are described in statistical terms by so called *Bernoulli Trials* which are parametric with respect to two probabilities p and $q = 1 - p$ (which have nothing to do with the scheduling probabilities above). The central limit theorem for this type of tests [GS97, Thm 9.2] gives us an estimate for the probability that the experimental value $S_n = n \cdot X_n$ after n repetitions of the test will be in a certain interval $[a, b]$:

$$\lim_{n\to\infty} \mathbf{P}(a \leq S_n \leq b) = \frac{1}{\sqrt{2\pi}} \int_{a^*}^{b^*} \exp\left(\frac{-x^2}{2}\right)$$

where

$$a^* = \frac{a - np}{\sqrt{npq}} \quad \text{and} \quad b^* = \frac{b - np}{\sqrt{npq}}.$$

Unfortunately, the integral of the so called *standard normal density* on the right hand side of the above expression is not easy to obtain. In practical situations one has to resort to numerical methods or statistical tables, but it allows us — at least in principle — to say something about $\mathbf{P}(H)$.

Identifying S_n with $n \cdot p^n_A$ we can utilise the above expression to estimate the probability $\mathbf{P}(p_A(c) + \varepsilon/2 \leq p^n_A)$ which determines $\mathbf{P}(H)$. In order to do this we have to take:

$$\begin{aligned} a &= p_A(c) + \frac{\varepsilon}{2} \\ b &= \infty \\ p &= p_A(c) \\ q &= 1 - p_A(c). \end{aligned}$$

This allows us — in principle — to compute the probability:

$$\lim_{n\to\infty} \mathbf{P}\left(p_A(c) + \frac{\varepsilon}{2} \leq p_A^n(c) \leq \infty\right).$$

Approximating — as it is common in statistics — $\mathbf{P}(p_A(c) + \varepsilon/2 \leq p_A^n)$ by $\lim \mathbf{P}(p_A(c) + \varepsilon/2 \leq p_A^n)$ we get:

$$\begin{aligned}\mathbf{P}(H) &= 1 - \mathbf{P}\left(p_A(c) + \frac{\varepsilon}{2} \leq p_A^n(c)\right)\\ &\approx 1 - \lim_{n\to\infty} \mathbf{P}\left(p_A(c) + \frac{\varepsilon}{2} \leq p_A^n(c)\right)\\ &= 1 - \int_{a_0}^{\infty} \exp\left(\frac{-x^2}{2}\right)\end{aligned}$$

with

$$a_0 = \frac{n\varepsilon}{2}\frac{1}{\sqrt{npq}} = \frac{\varepsilon\sqrt{n}}{2\sqrt{pq}} = \frac{\varepsilon\sqrt{n}}{2\sqrt{p_A(c)(1-p_A(c))}}.$$

We see that the only way to increase the probability $\mathbf{P}(H)$, i.e. the confidence that we formulate the right hypothesis about the identity of X, is by minimising the integral. In order to do this we have to increase the lower bound a_0 of the integral. This can be achieved — as one would expect — by increasing the number n of experiments.

We can also see that for a smaller ε we have to perform more tests n to reach the same level of confidence, $\mathbf{P}(H)$: The smaller n the harder it is to distinguish A and B experimentally. Note that for $\varepsilon = 0$, the probability of correctly guessing which of the agents A and B is in the black box is $\frac{1}{2}$, which is the best blind guess we can make anyway. In other words: for $\varepsilon = 0$ we cannot distinguish between A and B.

Example 8. Consider the agents in Example 5. The problem is to determine from the experimentally obtained approximation of the observables $\mathcal{O}_n\left(\frac{1}{2}:X \parallel \frac{1}{2}:C\right)$ for $X = A$ or $X = B$ the true identity of X. If, for example, X is actually agent A and if we concentrate on the constraint $c \sqcup d \sqcup e$ we have

$$\varepsilon = \frac{1}{12} \text{ and } p = p_A(c \sqcup d \sqcup e) = \frac{7}{12}$$

The probability $\mathbf{P}(H)$ to formulate the correct hypothesis H depends on the lower bound a_0 of the above integral, i.e. the normal distribution $N(a_0, \infty)$:

$$\mathbf{P}(H) = 1 - \int_{a_0(n)}^{\infty} \exp\left(\frac{-x^2}{2}\right) = 1 - N(a_0, \infty).$$

The bound a_0 in turn depends on the number n of experiments we perform. The value of a_0 for 9 tests is:

$$a_0(9) = \frac{\sqrt{9}}{24}\frac{1}{\sqrt{\frac{7}{12} - (\frac{7}{12})^2}} = \frac{1}{8}\frac{12}{\sqrt{35}} = \frac{3}{\sqrt{140}} \approx 0.25355$$

while for 144 tests we get:

$$a_0(144) = \frac{\sqrt{144}}{24} \frac{1}{\sqrt{\frac{7}{12} - (\frac{7}{12})^2}} = \frac{1}{2} \frac{12}{\sqrt{35}} = \frac{6}{\sqrt{35}} \approx 1.0142$$

In other words, if we repeat the execution of $\frac{1}{2} : X \parallel \frac{1}{2} : C$ exactly 9 times, the probability of formulating a correct hypothesis H about the identity of X is about (using a normal distribution table, e.g. [GS97, p499]):

$$\mathbf{P}(H) = 1 - \int_{0.25}^{\infty} \exp\left(\frac{-x^2}{2}\right) \approx 0.5987,$$

but if we perform 144 test our confidence level will rise to

$$\mathbf{P}(H) = 1 - \int_{1.0}^{\infty} \exp\left(\frac{-x^2}{2}\right) \approx 0.8413.$$

For 9 tests the hypothesis formulated will be right with an about 60% chance, while for 144 tests it will be correct with about 85%.

3 Process Algebra Formulation of Noninterference

3.1 Probabilistic Process Algebra

Process algebras are specification languages (see, e.g., [BW90,BPS01]) that describe the behaviour of concurrent systems through actions, which in our setting are syntactically divided into output actions and input actions, and through algebraic operators, which in our setting are enriched with probabilistic information (see, e.g., [BBS95]). Here we consider a slight variant of the probabilistic process algebra introduced in [BA03] (a core algebra of the reacher calculus EMPA$_{gr}$ [BB00]). The algebraic model of a system communicates with the environment through its inputs and outputs and performs internal computations through special, unobservable actions, termed τ actions. Formally, we denote with *AType* the set of visible action types, ranged over by a, b, ... For each visible action type a, we distinguish the output action a and the input action a_*. The complete set of actions, termed *Act* and ranged over by π, π', ..., contains the input actions and the output actions with type in *AType* and the action τ. The set $\mathcal{L}$ of process terms, ranged over by P, Q, ..., is generated by the syntax:

$$P ::= \underline{0} \mid \pi.P \mid P +^p P \mid P \parallel^p_S P \mid P\backslash L \mid P/^p_a \mid A$$

where $S, L \subseteq$ *AType*, $a \in$ *AType*, and $p \in]0,1[$. $\underline{0}$ expresses the null, deadlocked term[1], and ., $+^p$, $\parallel^p_S$, $\backslash L$, and $/^p_a$ denote the prefix, alternative, parallel, restriction, and hiding operators, respectively. Constants A are used to specify recursive systems. In particular, we assume a set of constants defining equations of the form $A \stackrel{\Delta}{=} P$ to be given. In the rest of the paper, we restrict ourselves to the set $\mathcal{G}$ of finite state, closed, guarded terms of $\mathcal{L}$, which we call processes [Mil89].

[1] We omit $\underline{0}$ when it is clear from the context.

Now, we informally describe the algebraic operators and the probabilistic model through an example. The reader interested in details and proofs should refer to [ABG03].

Example 9. Consider the following abstraction of the Automatic Teller Machine interacting with a client (cf. Example 1 in Section 2.2):

$$Client \parallel_S^p ATM.$$

The communication interface between processes *Client* and ATM, defined by set $S = \{insert\_pin, cash, fail\}$, says that the two processes (i) interact by synchronously executing actions of type in S, and (ii) asynchronously and independently execute each other local action. Probability p is the parameter of a probabilistic scheduler that, in each system state, decides which of the two processes must be scheduled, i.e. *Client* with probability p and ATM with probability $1 - p$.

Now, let us detail each component in isolation. Process *Client* repeatedly tries to insert a pin until the right number allows it to withdraw the cash:

$$Client \triangleq insert\_pin.Client' +^q \tau.Client.$$

The alternative choice operator "$\_ +^q \_$" says that process *Client* can either insert a pin (output action *insert_pin*) with probability q, and afterwards behaving as process *Client*′, or stay idle (action τ) with probability $1 - q$, and afterwards behaving as the same process *Client*. The actions *insert_pin* and τ follow the *generative* model of probabilities [GSS95], which is the same model adopted by PCCP (cf. Section 2). In essence, the process autonomously decides, on the basis of a probability distribution (guided by parameter q), which action will be executed and how to behave after such an event.

$$Client' \triangleq cash_*.\underline{0} +^{q'} fail_*.Client.$$

Process *Client*′ waits for the answer provided by the environment, i.e., it can either withdraw cash in case the pin number is right (input action $cash_*$), and afterwards stopping its execution (see the null term $\underline{0}$), or receive an unsuccessful message (input action $fail_*$), and afterwards behaving as process *Client* again. In practice, process *Client*′ internally reacts to the choice of the action type, *cash* or *fail*, performed by the environment (i.e., the machine). Formally, the input actions $fail_*$ and $cash_*$ follow the *reactive* model of probabilities [GSS95]. In particular, if the machine decides to communicate the action of type *fail*, then the client performs with probability 1 the unique input action of that type, which leads to process *Client*. Similarly, if the machine outputs the action of type *cash*, then the client chooses the input action $cash_*$ and then stops its execution. As a consequence of such a behaviour, parameter q' is not considered or, equivalently, from the viewpoint of process *Client*′ in isolation, the choice between such actions is purely nondeterministic, because their execution is entirely guided by the external environment.

Process ATM, instead, is ready to accept new incoming pins or it stays idle:

$$ATM \triangleq (insert\_pin_*.fail.ATM +^r insert\_pin_*.cash.ATM) +^{r'} \tau.ATM.$$

The two actions $insert\_pin_*$ model the internal reaction of process ATM to the choice of the action type $insert\_pin$ performed by its environment (i.e., the client). Such a reaction is guided by a probability distribution associated with the input actions of type $insert\_pin$ that process ATM can perform. More precisely, whenever the action type $insert\_pin$ is chosen by the client, process ATM reacts by choosing either the first action $insert\_pin_*$ with probability r and then refusing the pin (output action $fail$), or the second action $insert\_pin_*$ with probability $1 - r$ and then delivering cash (output action $cash$). Alternatively, if process ATM is not accepting pins from the environment, the internal action τ is repeatedly executed to model the idle periods of the machine. The choice between the input actions $insert\_pin_*$ and such an internal event is nondeterministic (parameter r' is not considered), because the execution of an action $insert\_pin_*$ is entirely guided by the external environment.

According to the considerations above, the processes interact in the composed system as follows. In the initial state of our example, the system executes a move of process $Client$ with probability p: it executes either the internal move τ with probability $p \cdot (1-q)$, or the synchronising move $insert\_pin$ with probability $p \cdot q$ (with probability $p \cdot q \cdot r$ it executes an $insert\_pin$ action synchronised with the first input action of process ATM and with probability $p \cdot q \cdot (1-r)$ an $insert\_pin$ action synchronised with the second input action of process ATM). Note that the result of a synchronisation between an output action $insert\_pin$ and an input action $insert\_pin_*$ is again an output action of type $insert\_pin$ (similarly as in CSP [Hoa85]). On the other hand, the system may schedule with probability $1 - p$ process ATM by executing its internal action τ, which gets the *entire* probability $1 - p$ associated to process ATM. Afterwards, if, e.g., the winning action is the action $insert\_pin$ leading to term $Client' \parallel^p_S cash.ATM$, then the system executes the synchronising action $cash$ with probability 1, because it is the unique action that can be performed by the composed system. In particular, note that action $fail_*$ of process $Client'$ is blocked, because the environment of process $Client'$, represented by process $cash.ATM$, is not available to output a corresponding output action $fail$. In Figure 8 we report the labeled transition systems that are associated with processes $Client$ and ATM in isolation and with the composed system.

The example above emphasises some features of the probabilistic process algebra that we now describe in more detail.

As far as the CSP-like communication policy is concerned, in any binary synchronisation at most one output action can be involved and, in such a case, the result is an output action of the same type. Instead, in case two input actions of type a synchronise, then the result is again an input action of type a. We recall that the actions belonging to the communication interface are constrained to synchronise, while all the other actions are locally and independently executed by the processes that compose the system.

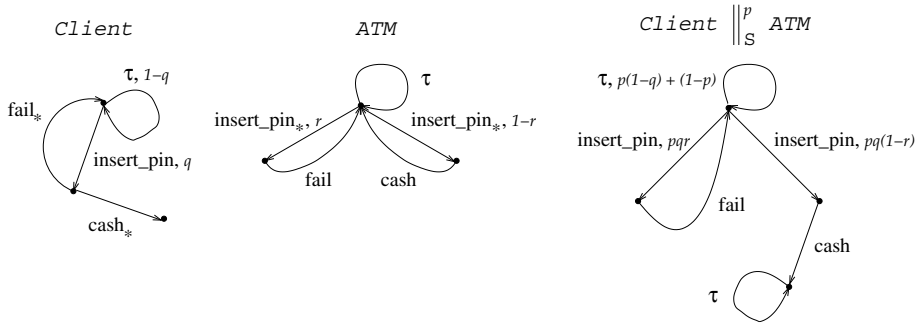


Fig. 8. Labeled transition systems associated to different process terms. Transitions are labeled with an action and a probability, which is equal to 1 if omitted

As far as the probability model is concerned, we have seen that output and internal actions follow the generative model, while input actions follow the reactive model. Probabilistic choices among output/internal actions or among input actions of the same type are fully probabilistic, while in each other case the choice is purely nondeterministic. This is because input actions are underspecified, in the sense that their execution is guided by the environment behaviour. Hence, the parameters that probabilistically guide the choices come into play if and only if a probabilistic choice is really to be performed. Moreover, Example 9 has emphasised the following behaviors of the parallel operator:

- In case the execution of some output actions of P is prevented in $P \|^p_S Q$ ($P \backslash L$), the probabilities of executing the remaining output/internal actions of P are proportionally redistributed (similarly for Q). That is a standard approach when restricting actions in the generative model of probabilities [GSS95], as also seen in case of PCCP (cf. Section 2).
- In case of synchronising output actions a of P in $P \|^p_S Q$, their probability is distributed among the multiple actions a obtained by synchronising with input actions a_* executable by Q, according to the probability the actions a_* are chosen in Q.

As a consequence of the policies specified above, we point out that in each system state of a process term, the sum of the probabilities of output and internal actions (input actions of a given type a), if there are any, is always equal to 1.

Now, we informally describe the behaviour of the hiding operator, which is needed to specify security properties. The hiding operator $P/^p_a$ turns output and input actions of type a into actions τ, by changing the probabilities according to the following rules.

- As far as output/internal actions executable by $P/^p_a$ are concerned, we distinguish the following cases:
  1. If P enables both some output/internal actions and some input actions a_*, then $P/^p_a$ chooses an action τ (obtained by hiding an action a_* of P) with probability p and an output/internal action previously enabled in P with probability $1 - p$. Such a rule guarantees that the hiding operator does not introduce nondeterminism among actions that follow the generative model of probability.

2. If either P does not enable output/internal actions, or P does not enable input actions a_*, then in $P/_a^p$ parameter p is not considered.

– As far as input actions are concerned, $P/_a^p$ enables the same input actions (with the same probability distribution) of type $b \neq a$ enabled in P.

Example 10. Consider process $P \stackrel{\Delta}{=} a_* +^{q'} (b +^q c)$, where the choice among a_* and the output actions is purely nondeterministic (parameter q' is not considered). The semantics of $P/_a^p$, which corresponds to process $\tau +^p (b +^q c)$, is a probabilistic choice between τ, executed with probability p, and the actions b and c, executed with probability $(1-p) \cdot q$ and $(1-p) \cdot (1-q)$, respectively. Hence, parameter p expresses the probability that the action τ obtained by hiding the input action a_* of P is executed with respect to the output actions previously enabled by P.

A goal of the hiding operator consists of turning open systems (i.e., systems enabling reactive choices due to input actions) into fully specified systems (i.e., fully generative systems, which do not include nondeterministic behaviours). In particular, the hiding operator resolves all the nondeterministic choices due to possible interactions with the environment by turning them into probabilistic choices. Intuitively, the effect of hiding an input action a_* corresponds to the execution of a synchronisation between a_* and an output action a offered by the environment. Such an interaction gives rise to an internal action τ whose probability distribution depends on parameter p of the hiding operator. When analysing security properties that employ the hiding operator, we will show that the low-level behaviour of a secure system does not depend on the choice of parameter p.

In the rest of the paper we use the following abbreviations. We assume parameter p to be equal to $\frac{1}{2}$ whenever it is omitted from any probabilistic operator. Moreover, when it is clear from the context, we use the abbreviation P/S, with $S = \{a_1, \ldots, a_n\} \subseteq \mathit{AType}$, to denote the expression $P/_{a_1} \ldots /_{a_n}$.

3.2 Operational Semantics and Equivalence

In this section, we provide a brief formal presentation of the semantics of the calculus. The reader not interested in such details can skip the rest of the section and proceed with the description of the security model.

The operational semantics of the probabilistic process algebra is given by the labeled transition system $(\mathcal{G}, \mathit{Act}, T)$, whose states are process terms and the transition relation T is the least multiset satisfying the operational rules reported in Table 3 and in Table 4. For a formal presentation of the semantics rules, the reader should refer to [BA03,ABG03], while here we just discuss some general aspects.

As far as the notation is concerned, we denote with RAct and GAct the sets of input actions, termed reactive actions, and of output and internal actions, termed generative actions, respectively. Then, we use the abbreviations $P \stackrel{\pi}{\longrightarrow}$ to stand for $\exists p, P' : P \stackrel{\pi, p}{\longrightarrow} P'$, denoting that P can execute action π with

probability p and then behave as P', and $P \xrightarrow{G}$, with $G \subseteq GAct$, to stand for $\exists a \in G : P \xrightarrow{a}$, meaning that P can execute a generative action belonging to set G.

As far as the rules for $P +^p Q$ and $P \parallel_S^p Q$ are concerned, note that in addition to the reported rules, which refer to the local moves of the left-hand process P, we also consider the symmetric rules taking into account the local moves of the right-hand process Q. Such symmetric rules are obtained by exchanging the roles of terms P and Q in the premises and by replacing p with $1 - p$ in the label of the derived transitions. Moreover, we also point out that for both operators, parameter p comes into play if and only if a probabilistic choice between P and Q is really to be performed. For instance, in case of the alternative choice operator, if P enables at least a generative action and Q does not, then $P +^p Q$ performs a generative transition of P with probability 1. Otherwise, if both P and Q enable some generative actions, then $P +^p Q$ performs a generative transition of P with probability p.

Two important remarks are in order in case of the parallel operator. On the one hand, if both P and Q can execute some synchronising actions a_* in $P \parallel_S^p Q$, then the composed system can execute some actions a_*: the probability of each action a_* executable by $P \parallel_S^p Q$ is the product of the probabilities of the two actions a_* (one of P and one of Q) that are involved in the synchronisation. On the other hand, as also explained in the previous section, when considering $P \parallel_S^p Q$ we must pay attention to the computation of the probability distribution of its generative actions, whose overall probability must sum up to 1. To this aim, in semantics rules we employ the function $\nu_P(G_{S,Q}) : \mathcal{P}(AType \cup \{\tau\}) \longrightarrow]0, 1]$, which computes the sum of the probabilities of the generative transitions of P (executable by $P \parallel_S^p Q$) whose type belongs to set $G_{S,Q} \subseteq AType \cup \{\tau\}$. In particular, set $G_{S,Q} = \{a \in AType \cup \{\tau\} \mid a \notin S \vee (a \in S \wedge Q \xrightarrow{a_*})\}$ contains the action types not belonging to the synchronisation set S and the action types belonging to S for which an input action of Q can be performed. Hence, $\nu_P(G_{S,Q})$ computes the aggregate probability of the generative transitions of P that can be executed by $P \parallel_S^p Q$ and can be used to normalise the probabilities of the generative transitions of P.

Finally, note that the tables omit the rules for the restriction operator. This is because it can be easily derived from the parallel operator. Indeed, we have that $P \backslash L$ corresponds to process $P \parallel_L \underline{0}$.

Since the security model we are going to present is based on the semantics of processes (i.e., the security check considers the program behaviour), we need an equivalence relation allowing for a comparison among the observable behaviours of different systems. To this aim, we resort to a probabilistic variant of the weak bisimulation [BH97], which abstracts away from τ actions and is able to identify deadlock. More precisely, such a relation, termed $\approx_{\mathrm{PB}}$, is a probabilistic extension of the nondeterministic weak bisimulation ($\approx_{\mathrm{B}}$) of [Mil89]. In essence, $\approx_{\mathrm{PB}}$ replaces the classical weak transitions of $\approx_{\mathrm{B}}$ by the probability of reaching classes of equivalent states. The notion of weak probabilistic bisimulation is based on the following definitions (for more details, see [ABG03]). We use a function

Table 3. Operational semantics (part I)

$$\pi.P \xrightarrow{\pi,1} P$$

$$\frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*}}{P +^p Q \xrightarrow{a_*,p\cdot q} P'} \qquad \frac{P \xrightarrow{a_*,q} P' \quad Q \not\xrightarrow{a_*}}{P +^p Q \xrightarrow{a_*,q} P'}$$

$$\frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{GAct}}{P +^p Q \xrightarrow{a,p\cdot q} P'} \qquad \frac{P \xrightarrow{a,q} P' \quad Q \not\xrightarrow{GAct}}{P +^p Q \xrightarrow{a,q} P'}$$

$$\frac{P \xrightarrow{a_*,q} P' \quad P \xrightarrow{GAct}}{P/^p_a \xrightarrow{\tau,p\cdot q} P'/^p_a} \qquad \frac{P \xrightarrow{a_*,q} P' \quad P \not\xrightarrow{GAct}}{P/^p_a \xrightarrow{\tau,q} P'/^p_a}$$

$$\frac{P \xrightarrow{b_*,q} P'}{P/^p_a \xrightarrow{b_*,q} P'/^p_a} \; a \neq b$$

$$\frac{P \xrightarrow{b,q} P' \quad P \xrightarrow{a_*}}{P/^p_a \xrightarrow{b,(1-p)\cdot q} P'/^p_a} \; a \neq b \qquad \frac{P \xrightarrow{a,q} P' \quad P \xrightarrow{a_*}}{P/^p_a \xrightarrow{\tau,(1-p)\cdot q} P'/^p_a}$$

$$\frac{P \xrightarrow{b,q} P' \quad P \not\xrightarrow{a_*}}{P/^p_a \xrightarrow{b,q} P'/^p_a} \; a \neq b \qquad \frac{P \xrightarrow{a,q} P' \quad P \not\xrightarrow{a_*}}{P/^p_a \xrightarrow{\tau,q} P'/^p_a}$$

$$\frac{P \xrightarrow{\pi,q} P'}{A \xrightarrow{\pi,q} P'} \text{ if } A \triangleq P$$

Prob such that $Prob(P, a_*, C)$ denotes the aggregate probability of going from P to a term in the class (of equivalent terms) C by executing an action a_*. Moreover, $Prob(P, \tau^* a, C)$ expresses the aggregate probability of going from P to a term in the equivalence class C via sequences of the form $\tau^* a$ (if $a \neq \tau$) or τ^* (if $a = \tau$). Formally:

$Prob(P, \tau^* a, C) =$

$$\begin{cases} 1 & \text{if } a = \tau \wedge P \in C \\ \sum_{Q \in \mathcal{G}} Prob(P, \tau, Q) \cdot Prob(Q, \tau^*, C) & \text{if } a = \tau \wedge P \notin C \\ \sum_{Q \in \mathcal{G}} Prob(P, \tau, Q) \cdot Prob(Q, \tau^* a, C) + Prob(P, a, C) & \text{if } a \neq \tau. \end{cases}$$

Definition 6. *An equivalence relation $R \subseteq \mathcal{G} \times \mathcal{G}$ is a weak probabilistic bisimulation if and only if, whenever $(P, Q) \in R$, then for all $C \in \mathcal{G}/R$:*

- $Prob(P, \tau^* a, C) = Prob(Q, \tau^* a, C)$ $\forall a \in GAct$
- $Prob(P, a_*, C) = Prob(Q, a_*, C)$ $\forall a_* \in RAct$.

Table 4. Operational semantics (part II)

$$\frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*}}{P \parallel^p_S Q \xrightarrow{a_*,p\cdot q} P' \parallel^p_S Q} \; a \notin S \qquad \frac{P \xrightarrow{a_*,q} P' \quad Q \not\xrightarrow{a_*}}{P \parallel^p_S Q \xrightarrow{a_*,q} P' \parallel^p_S Q} \; a \notin S$$

$$\frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*,q'} Q'}{P \parallel^p_S Q \xrightarrow{a_*,q\cdot q'} P' \parallel^p_S Q'} \; a \in S$$

$$\frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{G_{S,P}}}{P \parallel^p_S Q \xrightarrow{a,p\cdot q/\nu_P(G_{S,Q})} P' \parallel^p_S Q} \; a \notin S$$

$$\frac{P \xrightarrow{a,q} P' \quad Q \not\xrightarrow{G_{S,P}}}{P \parallel^p_S Q \xrightarrow{a,q/\nu_P(G_{S,Q})} P' \parallel^p_S Q} \; a \notin S$$

$$\frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \xrightarrow{G_{S,P}}}{P \parallel^p_S Q \xrightarrow{a,p\cdot q'\cdot q/\nu_P(G_{S,Q})} P' \parallel^p_S Q'} \; a \in S$$

$$\frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \not\xrightarrow{G_{S,P}}}{P \parallel^p_S Q \xrightarrow{a,q'\cdot q/\nu_P(G_{S,Q})} P' \parallel^p_S Q'} \; a \in S$$

Two terms $P, Q \in \mathcal{G}$ are weakly probabilistically bisimulation equivalent, denoted $P \approx_{\mathrm{PB}} Q$, if there exists a weak probabilistic bisimulation R containing the pair (P, Q).

Note that such a definition requires two equivalent terms to be strongly equivalent in case of reactive actions and weakly equivalent in case of generative actions. This is because τ is a generative action, therefore the computation of the probability of executing a mixed trace of generative/reactive actions (like, e.g., $\tau^* a_*$) does not actually make sense.

Example 11. Consider the processes $P \triangleq a +^{\frac{1}{2}} b$ and $Q \triangleq \tau.Q +^{\frac{1}{3}} (a +^{\frac{1}{2}} b)$, which, from an external observer viewpoint, behave the same since they execute either an output action a or an output action b with equal probabilities. We now want to formally verify such an intuition, i.e. we show that P and Q are weakly probabilistically bisimulation equivalent. Let R be the relation that considers the classes $\{C, [\underline{0}]\}$, where $C = \{P, Q\}$ and $[\underline{0}] = \{\underline{0}\}$. The only interesting case is given by $Prob(P, \tau^*\pi, [\underline{0}]) = \frac{1}{2}$, where $\pi \in \{a, b\}$. In order to compute the probability $Prob(Q, \tau^*\pi, [\underline{0}])$ we must consider that Q can execute an arbitrary number of times the action τ before reaching state $\underline{0}$ via an action a (b). To this aim, we redistribute the probability $\frac{1}{3}$ associated with the outgoing internal tran-

sition of Q among the other outgoing transitions of Q. Formally, by applying the definition of function $Prob$, we obtain $Prob(Q, \tau^* a, [\underline{0}]) = \frac{1}{3} \cdot Prob(Q, \tau^* a, [\underline{0}]) + \frac{1}{3}$, from which we derive $Prob(Q, \tau^* a, [\underline{0}]) = \frac{1}{2}$ (similarly for b). Hence, R is a weak probabilistic bisimulation and $P \approx_{\rm PB} Q$.

3.3 Probabilistic Noninterference

Probabilistic noninterference extends the classical, possibilistic definition of noninterference by providing the means for:

1. capturing those covert channels that are not observable in a purely nondeterministic setting, and
2. measuring the information leakage in terms of probability of observing the related covert channel.

In this section, we show how to formalise probabilistic noninterference in our process algebraic framework, while in the next one we extend the same approach in order to deal with the problem of giving a quantitative estimation of the information leakage.

As usual in security models, in our process algebraic framework we distinguish among high-level visible actions and low-level visible actions by defining two disjoint sets $AType_H$ of high-level types and $AType_L$ of low-level types, which form a covering of $AType$, such that the output action a and the input action a_* are high- (low-) level actions if $a \in AType_H$ ($a \in AType_L$). Usually, we use $l, l', \ldots$ to denote low-level types and $h, h', \ldots$ to denote high-level types. Then, in such a setting, we provide a semantics-based approach to noninterference, i.e., an approach where different program behaviours are compared to analyse a security property. Roughly, we derive two models from the algebraic specification of the system at hand, and then check the semantic equivalence between such derived models. On the one hand, the definition of semantic equivalence between processes is based on the weak probabilistic bisimulation equivalence $\approx_{\rm PB}$. On the other hand, the choice of the sub-models to be compared depends on the definition of the security property. Here, we consider the noninterference property of [Ald01,ABG03], which in turn is the probabilistic version of the Strong Nondeterministic Noninterference property proposed in [FG95] to express the classical noninterference idea of [GM82]. In essence, in order to detect potential high-level interferences, we compare the low-level behaviours of the system model P that can be observed in two different scenarios differing in the high-level behaviours only. In the former scenario, P is isolated from the high-level environment, so that all its high-level interactions are prevented, while in the latter scenario, P interacts with any high-level user that enables all the high-level actions of P.

The definition of Probabilistic Noninterference, here termed PNI, is as follows. For the sake of conciseness, we denote with $h_1^P, \ldots, h_n^P$ the sequence (in alphabetic order) of high-level types that syntactically occur in the action prefix operators within P.

Definition 7. $P \in PNI \Leftrightarrow P \backslash AType_H \approx_{PB} P /^{p_1}_{h_1^P} \ldots /^{p_n}_{h_n^P} \ \forall p_1, \ldots, p_n \in]0, 1[$.

Such a formulation also defines the particular class of adversaries (high-level users) with respect to which the probabilistic noninterference property is parameterised. Formally, according to the *PNI* definition, we can argue as follows.

On the one hand, $P \backslash AType_H$ expresses the low-level view of the system in isolation (without high-level interactions with the environment), since all the high-level actions are prevented.

On the other hand, $P /^{p_1}_{h_1^P} \ldots /^{p_n}_{h_n^P} \forall p_1, \ldots, p_n \in]0,1[$, where all the high-level actions are hidden, expresses the low-level view of P in case all the high-level interactions with the environment are enabled. In this formula, the hiding operator models the behaviour of any high-level user H that allows all the high-level actions enabled by P to be executed. More precisely, H allows the high-level output actions of P (turned into internal τ actions) to be executed with the probability distribution chosen by P itself. On the other hand, H allows the high-level input actions of P (turned into internal τ actions) to be executed with a probability distribution chosen by H itself according to parameters $p_1, \ldots, p_n$.

The class of attackers considered by the *PNI* property, here called $\mathcal{A}_{PNI}$, contains active and memoryless high-level users. More precisely, they are active as they can affect the probabilistic behaviour of the system activities, and they are memoryless as they cannot alter their behaviour depending on the previous history. In particular, as stated by the hiding operators, the probability distributions for the high-level inputs are chosen a priori and do not change during the system execution.

Example 12. Consider a program that writes a low-level variable in two possible ways, only one being legal, and represented by the following system:

$$P \stackrel{\Delta}{=} \tau \,.\, (copy\_secret\_PIN +^{0.001} copy\_random\_value) +^{p} high \,.\, (copy\_secret\_PIN +^{0.5} copy\_random\_value).$$

If the high-level user interacts with the system (such a communication is modeled by the execution of the high-level action *high*), then the program assigns to the public variable either a confidential value (low-level action *copy_secret_PIN*) with probability 0.5 or a random value (low-level action *copy_random_value*) with equal probability 0.5. On the other hand, if the high-level user does not interfere, then the program performs an internal activity that leads to the execution of the illegal assignment with a negligible probability. The choice between the interaction with the high-level user and the internal action is left to the system, which performs it according to parameter p.

A nondeterministic approach to noninterference[2] does not reveal any covert channel. This is because independently of the high-level behaviour, the low-level view of the system is always the same. However, if an external observer considers the outcomes of repeated executions of the system, then the relative frequency of such outcomes reveals the high-level interference. Formally, we have that $P \backslash AType_H$ and $P / AType_H$ are not weakly probabilistically bisimulation equivalent. For instance, we have that $P \backslash AType_H$ performs the action

[2] [Ald02,ABG03] rephrase the approach of [FG95] in a nondeterministic simplification of our process algebra, thus obtaining the same security property taxonomy.

copy_secret_PIN (preceded by an invisible transition) with probability 0.001, while $P/AType_H$ executes the same observable action (preceded by an invisible transition) with probability $p \cdot 0.001 + (1-p) \cdot 0.5$. Therefore, the *PNI* property is more than enough to capture the probabilistic covert channel described above.

3.4 Approximate Noninterference

In this section, we show how the knowledge about the probabilistic behaviour of a system may help the modeler to give a quantitative estimation of each information leakage, thus overcoming the qualitative view according to which a system is or is not secure. More precisely, given a covert channel that is responsible for an illegal information flow (which, e.g., could be revealed also in the possibilistic setting), we can evaluate the effectiveness of such a covert channel, by measuring the probability for an external observer of detecting it.

From a practical standpoint, a quantitative (probabilistic) approach to information flow analysis is useful for the verification of the security level of systems for which probabilities play an important role. For instance, many problems can be solved by using deterministic algorithms that turn out to be secure and require exponential time. On the other hand, probabilistic algorithms are often implemented that solve the same problems in polynomial time (see, e.g., [CKV00,MR99]). In such a case, the price to pay for a computational gain is the possibility for the observer of detecting an illegal information flow. Because of such a possibility, a probabilistic algorithm cannot be secure in case we limit the information flow analysis to the nondeterministic case. Instead, if we resort to a probabilistic approach, we can formally prove that the same algorithm has an illegal information flow, which, however, occurs with probability close to 0 (see, e.g., [AG02]). Based on these considerations, we need a quantitative approach in order to estimate the difference between the non-secure system and a secure one.

In our process algebraic setting, we may try to analyse the labeled transition system underlying an algebraic specification, in order to compute the probability that an information flow (from high level to low level) really happens. Unfortunately, a solution to such a problem cannot be provided if the verification of the security properties depends on a behavioural equivalence relation like the weak probabilistic bisimulation considered in the previous sections. This is because any equivalence relation states whether or not two given transition systems behave exactly the same. From a security standpoint, such an approach simply provides a binary answer: the system suffers or does not suffer an information leakage. Hence, small fluctuations in the system behaviour cannot be tolerated. Instead, we need a relaxed relation, which cannot be an equivalence relation, allowing for *similar* processes to be related, where the term *similar* stands for "behave almost the same up to small fluctuations".

On the basis of the considerations above, we now introduce a quantitative notion of behavioural similarity for deciding if two probabilistic processes are confined or, more precisely, for measuring the distance between probabilistic transition systems.

Formally, we now introduce the definition of weak probabilistic bisimulation with ε-precision, which is a relaxed version of the weak probabilistic bisimulation $\approx_{\mathrm{PB}}$ presented in Section 3.2.

Definition 8. *A relation $R \subseteq \mathcal{G} \times \mathcal{G}$ is a weak probabilistic bisimulation with ε-precision, where $\varepsilon \in]0,1[$, if and only if, whenever $(P,Q) \in R$, then for all $C \in \mathcal{G}/R$:*

- $\mid Prob(P, \tau^* a, C) - Prob(Q, \tau^* a, C) \mid \leq \varepsilon \quad \forall a \in GAct$
- $\mid Prob(P, a_*, C) - Prob(Q, a_*, C) \mid \leq \varepsilon \quad \forall a_* \in RAct.$

We use the abbreviation $P \approx_{PB_\varepsilon} Q$ to denote that there exists a weak probabilistic bisimulation with ε-precision R containing the pair (P,Q); alternatively, we say that P (Q) is a ε-perturbation of Q (P). Note that $\approx_{PB_\varepsilon}$ is not a transitive relation and, therefore, it cannot be an equivalence relation.

Example 13. Let us consider the fully specified transition systems depicted in Figure 9, which enable generative transitions only. It is easy to see that they cannot be weakly probabilistically bisimulation equivalent according to the definition of $\approx_{PB}$. Indeed, we have that s_2 and s_4 belong to the same equivalence class, while s_0, s_1, and s_3 are in three separate classes, since they have different probabilities of reaching the class $[\underline{0}]$ of the null term by executing the sequence $\tau^* a$ $(\tau^* b)$. However, we can observe that the observable behaviours of such systems are almost the same up to a perturbation ε. More formally, if we tolerate a distance at most equal to ε, we can define a relation that is a weak probabilistic bisimulation with ε-precision as follows. First, we immediately obtain that s_1 and s_2 (s_4) are similar, i.e. they belong to the same class C. For the same reason, we have that s_0 is in C, since $Prob(s_0, \tau^* a, [\underline{0}]) = \frac{1}{2} \cdot (\frac{1}{2} + \varepsilon) + \frac{1}{4} = \frac{1}{2} + \frac{1}{2} \cdot \varepsilon$ (similarly, for b we obtain $\frac{1}{2} - \frac{1}{2} \cdot \varepsilon$). Finally, s_3 is in C too, since $Prob(s_3, \tau^* a, [\underline{0}]) = \varepsilon + \frac{1}{2} \cdot (1 - \varepsilon) = \frac{1}{2} + \frac{1}{2} \cdot \varepsilon$ and $Prob(s_3, \tau^* b, [\underline{0}]) = \frac{1}{2} \cdot (1 - \varepsilon) = \frac{1}{2} - \frac{1}{2} \cdot \varepsilon$. Therefore, we have obtained a weak probabilistic bisimulation with ε-precision including the pair (s_0, s_3), i.e. the two transition systems are a ε-perturbation of the same system.

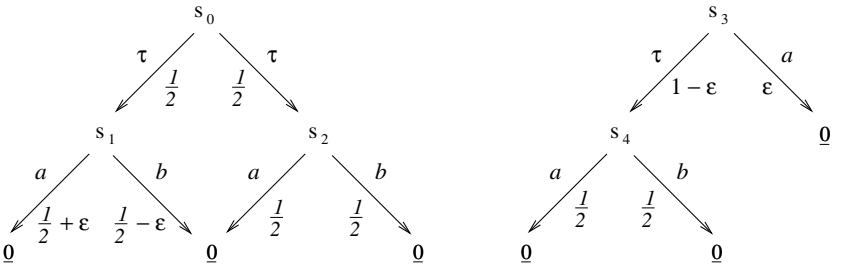


Fig. 9. Example of weak probabilistic bisimulation with ε-precision

3.5 Approximating *PNI*

The similarity relation can be used to approximate the noninterference property by simply replacing the equivalence relation in its formulation with such a similarity relation. In essence, instead of qualitatively asserting whether or not two sub-models of the system are equivalent, we just look at how much they differ. Since the sub-models to be compared express the low-level behaviour in case the system is isolated from the high environment and the low-level behaviour in case the system interacts with high users, respectively, an approximated noninterference property quantitatively states the capacity of a low-level observer of guessing the high environment behaviour by observing the system execution.

In our setting, the definition of process similarity is not parametric with respect to a specific set of adversaries (admissible spies, as termed in Section 2.4). Instead, the given security property is parameterised by a particular class of adversaries. Hence, security strictly depends on the definition of the property. In particular, here we show what happens when approximating the *PNI* property, which, as we have seen, is parameterised with respect to a particular class $\mathcal{A}_{PNI}$ of adversaries. In particular, if we replace in the definition of *PNI* the weak probabilistic bisimulation with the weak probabilistic bisimulation with ε-precision, we obtain a relaxed property that states if the behaviour of P in isolation is close (according to the *distance* ε) to that observed when P interacts with anyone of the high-level users in $\mathcal{A}_{PNI}$.

Example 14. Consider the system of Figure 10:

$$P \stackrel{\Delta}{=} h.l'.\underline{0} +^{p} \tau.(l.\underline{0} +^{q} h.l.\underline{0})$$

where it can be observed that:

- the left-hand component, which is chosen with probability p, is clearly nonsecure, since the execution of the action l' reveals to the low-level observer that the action h occurred;
- the right-hand component, which is chosen with probability $1-p$, is secure, since independently of the (probabilistic) high behaviour a low-level observer always sees the action l with probability 1.

We point out that the probabilistic information is not necessary to capture a covert channel in P, which is easily revealed as a "1-bit covert channel" by the nondeterministic counterpart of the *PNI* property [ABG03]. In other words, the probabilistic information described in P is not responsible for the information leakage. In spite of this, such an information turns out to be useful to analyse the security level of P. Indeed, the observation of the frequency of the possible outcomes of repeated executions of the system reveals that the behaviour of P is secure with probability $1-p$ and discloses an unwanted information flow with probability p. In practice, after a certain number, let us say n, of experiments during which the high-level user interacts with P, it turns out that the mean number of l' that have occurred is $n \cdot p$, while the mean number of l that have occurred is $n \cdot (1-p)$. Instead, after n experiments during which the high-level

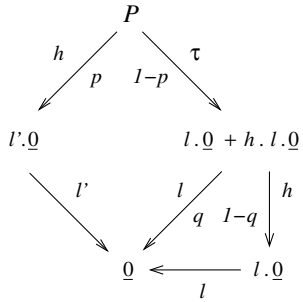


Fig. 10. Example of probabilistic information flow

user does not interact with P, it turns out that the number of l that have occurred is n. Obviously, by observing the relative frequencies "on the long run" of the observable results, we have that $P\backslash AType_H$ and $P/AType_H$ will differ by exactly a factor p. That means if an external observer executes the system (under one of the two scenarios) "infinitely often", then it can determine whether or not the high-level user was interfering. However, in a realistic scenario, after a finite number n of experiments and in case p is a value very close to 0, it is very hard for an external observer to understand whether or not the system was interacting with the high-level user. In such a case, the covert channel occurs with a negligible probability and P may be considered as a good approximation of a secure system.

The standard interpretation of probabilities as relative frequencies also helps to give an estimation of the covert channel capacity. Indeed, if we assume, e.g., that the system above is executed n times per week, then we can conclude that such a system suffers an information leakage equal to $n \cdot p$ bits per week, since that is (on average) the number of experiments that reveals the high-level user behaviour.

Now, we formally show how the weak probabilistic bisimulation with ε-precision is able to determine the security level of P. According to the *PNI* definition, we have $P\backslash AType_H \not\approx_{PB} P/AType_H$. However, $P\backslash AType_H$ is a p-perturbation of $P/AType_H$, since $P\backslash AType_H \approx_{PB} \tau.l.\underline{0} \approx_{PB} \tau.(l.\underline{0} +^q \tau.l.\underline{0}) \approx_{PB_p} \tau.l'.\underline{0} +^p \tau.(l.\underline{0} +^q \tau.l.\underline{0}) \approx_{PB} P/AType_H$. Therefore, the system can be considered secure enough as p tends to the value 0. Note that, according to the definition of weak probabilistic bisimulation with ε-precision, if p is less than the threshold ε, then the subsystem P' reached from $P/AType_H$ by executing the hidden high-level action h is simply disregarded, since it expresses a behaviour of the system reachable with a negligible probability. Therefore P' is not to be related with any corresponding behaviour of the system $P\backslash AType_H$.

Example 15. As another example, consider the following probabilistic process:

$$P \triangleq (l.\underline{0} +^p l.l'.\underline{0}) +^q l.h.l'.\underline{0}.$$

It is easy to see that P is not *PNI* secure. Formally, let us denote by C_1 the equivalence class of the null term $\underline{0}$ and by C_2 the equivalence class of

term $l'.\underline{0}$. On the one hand, we have $Prob(P\backslash AType_H, \tau^* l, C_1) = q \cdot p + 1 - q$ and $Prob(P\backslash AType_H, \tau^* l, C_2) = q \cdot (1-p)$. On the other hand, we have $Prob(P/AType_H, \tau^* l, C_1) = q \cdot p$ and $Prob(P/AType_H, \tau^* l, C_2) = 1 - q \cdot p$. Therefore, $|Prob(P\backslash AType_H, \tau^* l, C_1) - Prob(P/AType_H, \tau^* l, C_1)| = 1 - q = |Prob(P\backslash AType_H, \tau^* l, C_2) - Prob(P/AType_H, \tau^* l, C_2)|$, from which we derive that (i) process P does not satisfy the PNI property, and (ii) $P\backslash AType_H \approx_{PB_\varepsilon} P/AType_H$ if $q \geq 1 - \varepsilon$. Intuitively, if q is close to 1, then the low view of P, with or without the interaction with the high-level user, changes according to a small ε-fluctuation. While on the long run such a difference can be precisely identified, for a finite number of experiments $P\backslash AType_H$ and $P/AType_H$ turn out to behave almost the same. That means if we observe the low-level outcome of repeated executions of the system we are not able to notice the behaviour of the high-level user, since the high interference changes the frequency associated with each possible low-level outcome according to small, negligible fluctuations.

3.6 Statistical Interpretation

In a realistic scenario, an external observer makes a guess about the high environment behaviour after a certain number of tests (system executions). That means we need a formal way to measure the difference (by a finite number of experiments) between the low view of P in isolation, modeled by process $P\backslash AType_H$, and the low view of P interacting with any high user in $\mathcal{A}_{PNI}$, expressed by process $P/^{p_1}_{h_1^P} \ldots /^{p_n}_{h_n^P}$ for any sequence of probabilities $p_1, \ldots, p_n \in]0,1[$. The capability of the observer of revealing the difference between such processes expresses a measure of the effectiveness of the covert channel from high level to low level.

As an expected result, we can rephrase in our setting the same approach described in Section 2.5 to evaluate the confidence we can have in our hypothesis about the identity of a process after a finite number of experiments. We omit the technical part concerning the statistical methods behind such an approach (see Section 2.5) and we directly proceed with some clarifying examples.

Example 16. Consider the system:

$$\begin{aligned} P &\triangleq h_*.(l.\underline{0} +^{\frac{2}{3}} l'.\underline{0}) + (l.\underline{0} +^{\frac{7}{12}} l'.\underline{0}) \text{ such that} \\ P\backslash AType_H &\approx_{PB} (l.\underline{0} +^{\frac{7}{12}} l'.\underline{0}) \text{ and} \\ P/^p_h &\approx_{PB} \tau.(l.\underline{0} +^{\frac{2}{3}} l'.\underline{0}) +^p (l.\underline{0} +^{\frac{7}{12}} l'.\underline{0}). \end{aligned}$$

According to the low view of the system in isolation, expressed by term $P\backslash AType_H$, a low-level observer sees the action l with probability $\frac{7}{12}$ and the action l' with probability $\frac{5}{12}$. On the other hand, if P interacts with a high-level user that synchronises with the reactive action h_* with probability p, then the low view of the system changes. In particular, a low-level observer sees the action l with probability $\frac{7}{12} + \frac{1}{12} \cdot p$ and the action l' with probability $\frac{5}{12} - \frac{1}{12} \cdot p$. That means for $p \in]0,1[$ the probability of observing the action l varies in the range $]\frac{7}{12}, \frac{2}{3}[$ and the probability of observing the action l' is in the range $]\frac{1}{3}, \frac{5}{12}[$. As

a consequence, it turns out that $P/_h^p$ is a $\frac{1}{12}-$perturbation of $P\backslash AType_H$ for all $p \in]0,1[$. Formally, it is easy to verify that $P\backslash AType_H \approx_{PB\,\frac{1}{12}} P/_h^p$, for all $p \in]0,1[$.

An external low-level observer tries to distinguish the case in which P is isolated from the high environment from the case in which P interacts with a high-level user. To this purpose, he observes the relative frequencies of the low-level outcomes that derive from repeated executions of the system. After a number n of experiments, he formulates a hypothesis about the scenario in which P has been executed. The confidence he can have in such a hypothesis can be determined as reported in Section 2.5. In particular, we know that an upper bound for the distance between processes $P\backslash AType_H$ and $P/_h^p$ is $\varepsilon = \frac{1}{12}$. If we consider the scenario in which P is isolated from the high environment and we concentrate on the low-level outcome l (whose probability is equal to $\frac{7}{12}$), we obtain the same results shown in Example 8. More precisely, if we assume $n = 9$, we have that the hypothesis formulated by the low-level observer will be right with an about 60% chance, while for $n = 144$ it will be correct with about 85%.

Example 17. Now, let us consider again the same process P of Example 15. We want to estimate the confidence an external observer can have in a hypothesis about the high environment behaviour after a finite number n of experiments. To this purpose, let us assume $p = 0.5$ and $q = 0.99$. Such a scenario expresses the fact that the two possible behaviours (i.e. the single output l and the sequence $l.l'$) are chosen by the system with equal probabilities except for a small fluctuation due to scarce interferences by the high-level user. Formally, in $P\backslash AType_H$ the probability of observing the sequence $l.l'$ is equal to 0.495, while in $P/AType_H$ such a probability is equal to 0.505. Symmetrically, we can compute the probability of observing a single l, which is equal to 0.505 for $P\backslash AType_H$ and equal to 0.495 for $P/AType_H$. According to what we have shown in Example 15, the distance between such processes is $\varepsilon = 0.01$. Now, we assume that the high-level user is interacting with the system and we concentrate on the sequence of events $l.l'$. The probability $\mathbf{P}$ for an external low-level observer to identify the correct high environment behaviour depends on the number n of experiments. In particular, for $n = 10$ we have (cf. Section 2.5):

$$a_0(10) = \frac{10 \cdot 0.01}{2} \frac{1}{\sqrt{10 \cdot 0.505 \cdot 0.495}} \approx 0.03$$

and

$$\mathbf{P} = 1 - \int_{0.03}^{\infty} \exp\left(\frac{-x^2}{2}\right) \approx 0.512$$

Hence, for 10 tests the hypothesis that the observer formulates will be right with about 51%. Note that the probability of the best blind guess the observer can make is exactly 50%. We also emphasise that if we want such a probability to reach about 90%, then the external observer should execute about 16640 experiments.

3.7 The ATM Example

We present a simple but real example showing the need for a quantitative estimation of illegal information flows. In particular, we consider an Automatic Teller Machine (ATM), which gives cash if and only if the client inserts the unique, correct PIN number i (of m possible PINs) within a fixed number, say n, of attempts, after which the ATM retires the card:

$$ATM_k \triangleq insPIN_{i_*}.cash.ATM_1 + \sum_{j=1,j\neq i}^{m} insPIN_{j_*}.fail.ATM_{k+1} \quad 0 < k < n$$
$$ATM_n \triangleq insPIN_{i_*}.cash.ATM_1 + \sum_{j=1,j\neq i}^{m} insPIN_{j_*}.retire.ATM_1$$

An attacker that is in possession of the card (but not of the PIN) may try to illegally withdraw cash:

$$Spy \triangleq insPIN_1.Spy' +^{p_1} (insPIN_2.Spy' +^{p_2} \ldots)$$
$$Spy' \triangleq cash_*.spend.\underline{0} + fail_*.Spy + retire_*.flee.\underline{0}$$

We can assume that *cash* is the unique low-level action, since it expresses the tangible proof that a dishonest spy withdrew cash, while all the other events are considered to be high-level actions. If we take the composed system

$$ATMSys \triangleq ATM_1 \parallel_{\{cash,retire,fail,insPIN_i,\, i=1,\ldots,m\}} Spy$$

and check the nondeterministic counterpart of *PNI* [ABG03], we observe that the system is clearly non-secure. Indeed, if we hide the high-level actions, expressing the fact that the attacker interacts with the machine, then the action *cash* is observable. On the contrary, if we purge the system of the high-level actions, modeling the lack of any interaction between the machine and the attacker, then the action *cash* is not executable. Obviously, a purely nondeterministic approach captures the fact that an illegal behaviour can be observed in case the spy guesses the right PIN. In a realistic scenario, such an event is possible but negligible. For instance, assume that for any attempt the spy randomly samples a PIN value according to a uniform distribution, and take two realistic parameters, i.e. $m = 100000$ and $n = 3$. Then, denoted C the equivalence class of the null term, we have that $Prob(ATMSys/AType_H, \tau^* cash, C) \approx 0.00003$. Formally, if we employ the weak probabilistic bisimulation with ε-precision ($\varepsilon = 0.00003$), then the system turns out to satisfy the approximated *PNI* property. This is because the probability of observing the illegal cash leakage is considered to be negligible.

4 Related Work and Conclusion

In this paper, we surveyed two techniques for approximating noninterference properties, thus enriching the intuition behind the definition of probabilistic noninterference, which appeared in the literature to overcome the limitations of classical possibilistic approaches to information flow analysis. Initially, a formulation of probabilistic covert channel was proposed in [McL90,Gra90], and

later on in [Gra92] and in [GS92,SG95]. More recently, in [SS00] the same intuition has been rephrased in the setting of an imperative language with dynamic thread creation, where, as a novelty, a probabilistic notion of bisimulation is used to formalise a security condition. In [Smi01], a type system is presented that aims to ensure secure information flow in a simple multi threaded imperative programming language running under a uniform probabilistic scheduler. The same author also employs a definition of weak probabilistic bisimulation (inspired by [BH97]) in [Smi03].

In the first approach presented in this paper we have concentrated on a notion of observable behaviour for programs in the PCCP language, which corresponds to the probabilistic input/output observables. These can be described by probability distributions on the underlying space of constraints, and we used a vector norm to measure their similarity. By considering the observables of two processes executed in the context of a spy we were then able to measure their confinement. Different analyses can be constructed depending on the type of attacks we consider. For example, in [DHW03b,DHW02b] a control-flow analysis for the confinement property is presented which refers to *internal attacks*. This is the case where the attacker is part of the observed system and is therefore subject to the same scheduler as the host system. In another context one might be interested in *external attacks*, where the attacker is only allowed to observe the system from the outside and is thus scheduled in a different way, or one might impose other restrictions on the way a spy may observe the agents in question. In [DHW02a], an analysis is presented for the case of external attacks, which exploits information about the average store of an agent in some specified number of steps (the observation time).

In the second approach we described, the notion of observable behaviour for processes is formalised in the process algebraic calculus of [BA03], whose semantics is given in terms of a probabilistic version of the weak bisimulation equivalence [BH97]. In this setting, we have shown that the robustness of a system against a specified class of attackers (as defined by the probabilistic noninterference property) can be checked by following the same approach introduced in [FG95] in a purely nondeterministic framework. Along this line, in [ABG03] a complete taxonomy of probabilistic security properties is described. The expressiveness of the probabilistic process algebra and of the particular model of probability we adopted allow us to model and analyse real, complex systems. For example, in [AG02], a case study shows the adequacy of such an approach for analysing the security level (under any probabilistic adversary) of a probabilistic cryptographic protocol [MR99] implemented to achieve a fairness property.

In the literature, several papers propose a formal definition of approximated bisimilarity. For example, in [vBW01,DGJP99] different pseudometrics are introduced that quantify the similarity of the behavior of probabilistic transition systems that are not bisimilar. In particular, in [DGJP99] the authors consider a metric on partial labeled Markov chains, which are a generalization of the fully specified transition systems described in Sect. 3, in that for each state the sum of the probabilities of the outgoing transitions, if there are any, is less than (or

equal to) 1, while in our case such a sum always sums up to 1. Moreover, they extend the same approach to the weak bisimulation case in [DGJP02]. With respect to the notion of approximated weak probabilistic bisimulation $\approx_{PB_\varepsilon}$, the approximated equality introduced in [DGJP02] is compositional. On the other hand, $\approx_{PB_\varepsilon}$ allows systems that can have largely different possible behaviours to be related under the condition that such behaviours are observable with a negligible probability. Another approach to the approximation of bisimilarity has been recently proposed in [DHW03c,DHW03a], which extends the approach presented in this paper to probabilistic transition systems and is based on the definition of bisimulation via a linear operator and the use of an operator norm for measuring noninterference.

Acknowledgement

This work has been partially funded by Progetto MEFISTO (Metodi Formali per la Sicurezza e il Tempo) and the EU FET open projects SecSafe and Degas.

References

Ald01. A. Aldini. Probabilistic Information Flow in a Process Algebra. In Proc. of *12th Int. Conf. on Concurrency Theory (CONCUR'01)*, Springer LNCS 2154:152–168, 2001.

Ald02. A. Aldini. On the Extension of Non-interference with Probabilities. In Proc. of *WITS'02 – 2nd Workshop on Issues in the Theory of Security* (J. Guttman, Ed.), Portland, OR (USA), 2002.

ABG03. A. Aldini, M. Bravetti, and R. Gorrieri. A Process-algebraic Approach for the Analysis of Probabilistic Noninterference. *Journal of Computer Security*, to appear.

AG02. A. Aldini and R. Gorrieri. Security Analysis of a Probabilistic Non-repudiation Protocol. In Proc. of *2nd Joint Int. Workshop on Process Algebra and Performance Modelling, Probabilistic Methods in Verification (PAPM-ProbMiV'02)*, Springer LNCS 2399:17–36, 2002.

BW90. J.C.M. Baeten and W.P. Weijland. *"Process Algebra"*, Cambridge University Press, 1990.

BBS95. J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing Probabilistic Processes: ACP with Generative Probabilities. *Information and Computation* 121:234–255, 1995.

BH97. C. Baier, and H. Hermanns. Weak Bisimulation for Fully Probabilistic Processes. In Proc. of *9th Int. Conf. on Computer Aided Verification (CAV'97)*, Springer LNCS 1254:119–130, 1997.

BPS01. J.A. Bergstra, A. Ponse, and S.A. Smolka (Eds.) *Handbook of Process Algebra*, Elsevier Science Publishers B.V., Amsterdam, 2001.

Ber99. M. Bernardo. Theory and Application of Extended Markovian Process Algebra. *Ph.D. Thesis*, University of Bologna, Italy, 1999. `ftp://ftp.cs.unibo.it/pub/techreports/99-13.ps.gz`

BDG98. M. Bernardo, L. Donatiello, and R. Gorrieri. A Formal Approach to the Integration of Performance Aspects in the Modeling and Analysis of Concurrent Systems. *Information and Computation* 144(2):83–154, 1998.

Bil86. P. Billingsley. *Probability and Measure*. Wiley & Sons, New York, 2nd edition, 1986.

Bra02. M. Bravetti. Specification and Analysis of Stochastic Real-Time Systems. *Ph.D. Thesis*, University of Bologna (Italy), 2002. `http://www.cs.unibo.it/~bravetti`

BA03. M. Bravetti and A. Aldini. Discrete Time Generative-reactive Probabilistic Processes with Different Advancing Speeds. *Theoretical Computer Science* 290(1):355–406, 2003.

BB00. M. Bravetti and M. Bernardo. Compositional Asymmetric Cooperations for Process Algebras with Probabilities, Priorities, and Time. In Proc. of *1st Workshop on Models for Time-Critical Systems (MTCS'00)*, State College (PA), ENTCS 39(3), 2000.

vBW01. F. van Breugel, and J. Worrell. Towards Quantitative Verification of Probabilistic Systems (extended abstract). In Proc. of *28th Int. Colloquium on Automata, Languages and Programming (ICALP'01)*, Springer LNCS 2076:421–432, 2001.

BHK01. E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.) *Lectures on Formal Methods and Performance Analysis*, Springer LNCS 2090, 2001.

CKV00. C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography (extended abstract). In Proc. of *19th Symposium on Principles of Distributed Computing*, ACM Press, pp. 123–132, 2000.

CT02. M.C. Calzarossa and S. Tucci (Eds.) *Performance Evaluation of Complex Systems: Techniques and Tools*, Performance 2002 Tutorial Lectures, Springer LNCS 2459, 2002.

CHM02. D. Clark, S. Hunt, and P. Malacaria. Quantitative Analysis of Leakage of Confidential Data. In *QAPL 2001 - First International Workshop on Quantitative Aspects of Programming Laguages*, volume 59 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2002.

dDP95. F.S. de Boer, A. Di Pierro, and C. Palamidessi. Nondeterminism and Infinite Computations in Constraint Programming. *Theoretical Computer Science*, 151(1):37–78, 1995.

DGJP99. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for Labeled Markov Processes. In Proc. of *10th Int. Conf. on Concurrency Theory (CONCUR'99)*, Springer LNCS 1664:258–273, 1999.

DGJP02. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. The Metric Analogue of Weak Bisimulation for Probabilistic Processes. In Proc. of *17th Symposium on Logic in Computer Science (LICS)*, IEEE CS Press, pp. 413–422, 2002.

DHW01. A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic Confinement in a Declarative Framework. In *Declarative Programming – Selected Papers from AGP 2000 – La Havana, Cuba*, volume 48 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2001.

DHW02a. A. Di Pierro, C. Hankin, and H. Wiklicky. Analysing Approximate Confinement under Uniform Attacks. In Proc. of *SAS 2002 - 9th Int. Symposium on Static Analysis*, Springer LNCS 2477:310–326, 2002.

DHW02b. A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate Non-interference. In Proc. of *15th IEEE Computer Security Foundations Workshop*, pages 3–17, Cape Breton, Nova Scotia, Canada, 2002.

DHW03a. A. Di Pierro, , C. Hankin, and H. Wiklicky. Measuring the Confinement of Concurrent Probabilistic Systems. In Proc. of *WITS'03 – Workshop on Issues in the Theory of Security* (R. Gorrieri, Ed.), Warsaw, Poland, 2003. `http://www.dsi.unive.it/IFIPWG1_7/wits2003.html`.

DHW03b. A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate Non-interference. *Journal of Computer Security*, 2003. To appear.

DHW03c. A. Di Pierro, C. Hankin, and H. Wiklicky. Quantitative Relations and Approximate Process Equivalences. In Proc. of *14th Int. Conf. on Concurrency Theory (CONCUR'03)*, Lecture Notes in Computer Science, Springer Verlag, 2003. To appear.

DW98a. A. Di Pierro and H. Wiklicky. An Operational Semantics for Probabilistic Concurrent Constraint Programming. In Proc. of *ICCL'98 – Int. Conf. on Computer Languages*, P. Iyer, Y. Choo, and D. Schmidt, Eds., pp. 174–183, IEEE Computer Society Press, 1998.

DW98b. A. Di Pierro and H. Wiklicky. Probabilistic Concurrent Constraint Programming: Towards a Fully Abstract Model. In Proc. of *MFCS'98 – Mathematical Foundations of Computer Science*, L. Brim, J. Gruska, and J. Zlatuska, Eds., Springer LNCS 1450:446–455, 1998.

DW00. A. Di Pierro and H. Wiklicky. Quantitative Observables and Averages in Probabilistic Concurrent Constraint Programming. In *New Trends in Constraints – Selected Papers of the 1999 ERCIM/Compulog Workshop on Constraints*, K.R. Apt, T. Kakas, E. Monfroy, and F. Rossi, Eds., Springer LNCS 1865, 2000.

GM82. J.A. Goguen and J. Meseguer. Security Policy and Security Models. In Proc. of *Symposium on Security and Privacy (SSP'82)*, IEEE CS Press, pp. 11–20, 1982.

FG95. R. Focardi and R. Gorrieri. A Classification of Security Properties. *Journal of Computer Security* 3(1):5–33, 1995.

FG01. R. Focardi and R. Gorrieri (Eds.) *Foundations of Security Analysis and Design - Tutorial Lectures*, Springer LNCS 2171, 2001.

GSS95. R. J. van Glabbeek, S. A. Smolka, and B. Steffen. Reactive, Generative and Stratified Models of Probabilistic Processes. *Information and Computation* 121:59–80, 1995.

Gra90. J. W. Gray III. Probabilistic Interference. In Proc. of *Symposium on Security and Privacy (SSP'90)*, IEEE CS Press, pp. 170–179, 1990.

Gra92. J. W. Gray III. Toward a Mathematical Foundation for Information Flow Security. *Journal of Computer Security* 1:255–294, 1992.

GS92. J. W. Gray III and P. F. Syverson. A Logical Approach to Multilevel Security of Probabilistic Systems. In Proc. of *Symposium on Security and Privacy (SSP'92)*, IEEE CS Press, pp. 164–176, 1992.

GS97. C.M. Grinstead and J.L. Snell. *Introduction to Probability.* American Mathematical Society, Providence, Rhode Island, second revised edition, 1997.

HS95. P. Harrison and B. Strulo. Stochastic Process Algebra for Discrete Event Simulation. In *Quantitative Methods in Parallel Systems, ESPRIT Basic Research Series*, pp. 18–37, Springer, 1995.

HMT71. L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras (Part I).* North-Holland, 1971.

HHHMR94. H. Hermanns, U. Herzog, J. Hillston, V. Mertsiotakis, and M. Rettelbach. Stochastic Process Algebras: Integrating Qualitative and Quantitative Modelling. In *7th Conf. on Formal Description Techniques (FORTE'94)*, pp. 449–451, 1994.

Hil96. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

Hoa85. C. A. R. Hoare. *Communicating Sequential Processes*, Prentice Hall, 1985.

Koc95. P.C. Kocher. Cryptanalysis of Diffie-Hellman, RSA, DSS, and Other Cryptosystems Using Timing Attacks. In *Advances in Cryptology, CRYPTO'95: 15th Annual Int. Cryptology Conf.*, D. Coppersmith, Ed., Springer LNCS 963:171–183, 1995.

LMMS98. P.D. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. A Probabilistic Poly-time Framework for Protocol Analysis. In *ACM Conf. on Computer and Communications Security*, pp. 112-121, ACM Press, 1998.

MR99. O. Markowitch and Y. Roggeman. Probabilistic Non-Repudiation Without Trusted Third Party. In *2nd Conf. on Security in Communication Networks*, Amalfi, Italy, 1999.

McL90. J. McLean. Security Models and Information Flow. In Proc. of *Symposium on Security and Privacy (SSP'90)*, IEEE CS Press, pp. 180–189, 1990.

Mil89. R. Milner. *Communication and Concurrency*, Prentice Hall, 1989.

RMMG01. P.Y.A. Ryan, J. McLean, J. Millen, and V. Gligor. Non-interference: Who needs It? In Proc. of *14th Computer Security Foundations Workshop (CSFW'01)*, IEEE CS Press, pp. 237–238, 2001.

SS99. A. Sabelfeld and D. Sands. A Per Model of Secure Information Flow in Sequential Programs. In Proc. of *European Symp. on Programming (ESOP'99)*, Springer LNCS 1576:40–58, 1999.

SS00. A. Sabelfeld and D. Sands. Probabilistic Noninterference for Multi-threaded Programs. In Proc. of *13th Computer Security Foundations Workshop (CSFW'00)*, IEEE CS Press, pp. 200–215, 2000.

SM03. A. Sabelfeld and A.C. Myers. Language-Based Information Flow Security. *IEEE Journal on Selected Areas in Communications* 21(1):5–19, 2003.

SR90. V.A. Saraswat and M. Rinard. Concurrent constraint programming. In *Symposium on Principles of Programming Languages (POPL)*, pp. 232–245, ACM Press, 1990.

SRP91. V.A. Saraswat, M. Rinard, and P. Panangaden. Semantics foundations of concurrent constraint programming. In *Symposium on Principles of Programming Languages (POPL)*, pp. 333–353, ACM Press, 1991.

Sha99. J. Shao. *Mathematical Statistics*. Springer Texts in Statistics. Springer Verlag, 1999.

Smi01. G. Smith. A new Type System for Secure Information Flow. In Proc. of *14th Computer Security Foundations Workshop (CSFW'01)*, IEEE CS Press, pp. 115–125, 2001.

Smi03. G. Smith. Probabilistic Noninterference through Weak Probabilistic Bisimulation. In Proc. of *16th Computer Security Foundations Workshop (CSFW'03)*, IEEE CS Press, pp. 3–13, 2003.

SG95. P. Syverson and J. W. Gray III. The Epistemic Representation of Information Flow Security in Probabilistic Systems. In Proc. of *8th Computer Security Foundations Workshop (CSFW'95)*, IEEE CS Press, pp. 152–166, 1995.

The Key Establishment Problem

Carlo Blundo[1] and Paolo D'Arco[2]

[1] Dipartimento di Informatica ed Applicazioni
Università di Salerno, 84081 Baronissi (SA), Italy
carblu@dia.unisa.it
[2] Department of Combinatorics and Optimization
University of Waterloo, Waterloo Ontario, N2L 3G1, Canada
pdarco@cacr.math.uwaterloo.ca

Abstract. Key Establishment is one of the most intriguing, fascinating and deeply studied problems in Cryptography. In this paper we propose a *brief excursus* among ideas and techniques that during the last years have been applied in a variety of settings, in order to design suitable and often mathematically delightful protocols to solve this issue. The presentation uses a very simple language: it is basically an introduction to the subject. Hopefully, it is even self-contained. Formal proofs and details are omitted, but the interested reader can find them in the referred papers.

1 Introduction

Cryptography is currently spreadly used to protect digital communication and information processing. All the applications belonging to the so-called *electronic commerce* area and many information services offered by public or private organizations, are possible by the shrewd and refined use of cryptographic techniques. Roughly speaking, we could say that there is a *visible* digital world that most people experience every day, for example by using their personal computers at home for surfing the Internet, for accessing their bank accounts, or for buying goods from digital portals, which is built upon an underlying *hidden* world that exists to ensure that "everything goes fine" in the visible one. This hidden world is the world of Cryptography, an important aspect of which is the subject of these pages.

Around twenty years ago, people started foreseeing the large spectrum of possibilities for Cryptography[1]: indeed, the diffusion of public communication networks provides a very powerful media to exchange data, in order to solve common problems. Unfortunately, as long as users need to communicate to perform joint operations, several reasons can drive some of them to misbehaviors and unpredictable actions. Just to exemplify, if the community of users runs a digital protocol for the election of their representatives, it is not unrealistic to assume that some users can try either to falsify the result of the election or to find out for which candidate has voted a certain user. Therefore, some countermeasures must be taken.

[1] The newcomers to Cryptography are strongly encouraged to read Rivest's survey [106] and, for recent and future prospectives, Maurer's survey [92].

R. Focardi and R. Gorrieri (Eds.): FOSAD 2001/2002, LNCS 2946, pp. 44–90, 2004.

To get the picture, Cryptography can be described as *a collection of ideas and techniques* enabling the community of users to complete common tasks in such a way that misbehaviors from some of them are harmless. Basically, this goal is obtained by means of a *knowledge-gap* between users who wish to perform a certain task and users who, for several reasons, can decide to misbehave in arbitrary ways. Such gap assumes the form of secret information, referred to as *keys*, held by some honest users but not by dishonest ones.

The main question that comes up and we are going to investigate in the following pages is *how* can be established keys among groups of users of a network who wish to perform computations in a secure way.

1.1 Alice, Bob and the Secret Place

The first people that we meet in our excursus are *Alice* and *Bob*: every student who has given a look at a book on Cryptography in his life has surely met them at least once. The setting in which they belong to is the following: *Alice* and *Bob* need to privately communicate but they only share a public channel. Therefore, a third (bad) guy, *Eve*, could eavesdrop the communication. Hence, they decide to *encrypt* the messages they send to each other in order to be protected against *Eve*. Loosely speaking, an encryption scheme is a family of pairs of rules

$$\{(E_k(), D_k())\}_{k \in \mathcal{K}}$$

where $E_k()$ enables *Alice* to encrypt the messages she wishes to send to *Bob*, while $D_k()$ enables *Bob* to decrypt the encrypted messages received by Alice. More precisely, *Alice* computes and sends $c = E_k(m)$, where m is the message she would like Bob receives, and Bob computes $m = D_k(c) = D_k(E_k(m))$, and vice versa. Such process works if for each possible message m it results $m = D_k(E_k(m))$ (i.e., $D_k()$ is the inverse rule for $E_k()$). Alice and Bob choose the pair they want to use to protect the privacy of their communication *by choosing a value of* $k \in \mathcal{K}$, referred to as the *secret key*. $\mathcal{K}$ is the set of all possible secret keys.

For example, Alice and Bob can decide that the encryption rule consists of substituting every letter of the message with the one that follows in the alphabet, on which the message is defined, by 3 positions in cyclic order. Symmetrically, the decryption rule requires that every letter of the encrypted message is substituted by the letter 3 positions backwards in the alphabet. The secret key in this case is given by *the number* 3. *Eve* can even know that they encrypt and decrypt their communication by substituting the letters of the message with others of the same alphabet at a certain fixed distance, but since she does not know the *value* of this distance, she cannot decrypt any message.

Apart the security issue of the above strategy, historically used and known as the *Caesar's Cipher* [77], what is important in our investigation is: how do they fix a value for the secret key? To get started, we can say that they have a meeting in a *secret place*. It could seem trivial but it is what people have done for roughly two thousand years and in several settings they still do. As we will

see in the following, in many protocols, the so called *set up* phases, in which users get secret information, are the equivalent of the old meeting in a secret place.

From an historical point of view, it is not known neither if *Alice* and *Bob* have lived somewhere nor if they have ever had the need to privately communicate on a public channel: but for sure, they live in the cryptographic language and the problem they are presumed to manage is really one of the first that people have tried to solve with several techniques. About *Eve*, her identity is still more doubtful: sometimes she is called Oscar, sometimes Opponent, some others is called simply Adversary, but she/he does seem to exist, at least to justify Cryptography!

1.2 Keys in Cryptography

As we were saying before, keys, secret pieces of information belonging to a certain set, constitute the knowledge gap held by a group of users with respect to adversaries, by means of which the group can perform tasks in a secure way, like privately communicate. For example, the value of k that *Alice* and *Bob* choose in order to define a pair $(E_k(), D_k())$ among the set $\{(E_k(), D_k())\}_{k \in \mathcal{K}}$ is the knowledge gap that protects them against *Eve*.

To give an idea, some settings in which keys are used are:

- *Point-to-point private communications.* This is the setting we have considered before: two users, *Alice* and *Bob*, wish to privately communicate over a public channel. They use a secret key to encrypt and decrypt the messages they send to each other.
- *Multicast communications and conferencing.* Many users are involved in a private communication. This setting generalizes in several ways the previous one: it embraces private group communications, as well as multicast and broadcast communications, where a single source sends information to a certain subset of recipients, which changes from time to time.
- *Entity and Data Authentication.* Keys are used in protocols enabling one party to prove to another party his identity, i.e., the other party is convinced that the person that is speaking is the real one and not an adversary, or to guarantee the authenticity of a certain source of information.
- *Information Integrity Check.* Many cryptographic primitives, designed to check the integrity of information transmitted over insecure channels or stored in unreliable/breakable memories, use secret keys.

Moreover, keys can be classified according to their usage, life-time, and other features. Without going into details at the moment, keys can be:

- *Secret keys.* Used by users in symmetric cryptosystems and, more generally, with cryptographic primitives requiring one key.
- *Public keys.* Public known keys, usable by *all* the users of a network with a public key cryptosystem or a digital signature scheme.

- *Private keys.* The corresponding key of a certain public one, held and usable by a *single* user in a public key system, in order to decrypt or sign messages.
- *Session keys.* Used for a short period of time.
- *Master or Long Term keys.* Stored for a long time and often used to generate or derive session keys.

1.3 The Power of *Eve*

Cryptography concerns with design and analysis of protocols. A *multi-party protocol* is a well-defined sequence of steps that each party has to perform in order to obtain a fixed common goal. A *cryptographic protocol* is a multi-party protocol that keeps working (i.e., maintains its functionality) even in presence of an adversary who can simply listen the conversation that takes place among the users or that can coordinate the actions of some parties, in order to corrupt the output of the protocol or to obtain from the execution information that the protocol is not supposed to leak. A cryptographic protocol is *secure* if it is designed in such a way that no adversary can succeed in the above attempt. On the other hand, if an adversary can gain some advantage by listening or controlling some parties in deviating from the protocol, we say that he can *break* the protocol.

Just to exemplify the above concept in a concrete context, and with a certain degree of approximation, think about the private communication problem *Alice* and *Bob* have to solve: in that case, an encryption scheme (i.e., cryptographic protocol for private communication) is secure if, assuming that the only thing that *Eve* can do is to tap the channel, from the encrypted messages sent by the parties along the public channel, she cannot obtain *any partial information* about the real conversation.

Apart the strategy that an adversary can pursue in order to break a certain protocol, and the amount of information he can count on, a preliminary assumption that is done in order to study the security of protocols concerns with the *computational power* of the adversary: in other words, the amount of resources *Eve* can afford in order to succeed. This assumption leads to two different worlds in cryptography.

- *Computationally Secure Setting. Eve* is bounded. She can perform only feasible computations where, as usual in complexity theory, we refer with this term to procedures which require time and space upper bounded by a polynomial $P(n)$, where $n = |x|$ is the size of the instance x of the problem the procedure solves.
- *Unconditionally Secure Setting. Eve* is unbounded. She can use as much time and space as she needs: in this setting, even theoretical but infeasible computations are supposed to be real threats. A cryptographic protocol proved secure against such an adversary is usually referred to as *perfectly secure* because it is secure independently of the efforts of *Eve*.

Moreover, cryptographic protocols proved secure in the computational setting belong to two different families: in the first case, a protocol is showed to be resistant to all currently known and computationally feasible attack strategies.

Hence, the protocol is presumed to be secure modulo the non-existence of better strategies. In the second case, a protocol is "proved" secure because the existence of feasible strategies to break the security of the scheme implies the possibility of constructing a feasible procedure to solve some supposed to be infeasible mathematical problem. For example, factoring an integer n which is the product of two large primes, computing the discrete log in multiplicative groups of prime order, or computing roots of powers, are all presumed to be infeasible tasks for large value of n and suitable sizes of the groups. Hence, a proof of security in this case consists in showing that, if an efficient procedure to break a given protocol exists, then there exists an efficient procedure, say, to factorise a large integer n, product of two large primes, which is commonly believed to be false.

Therefore, we could say that in the first case the security is a sort of empiric security: the proof is given by means of a collection of arguments showing how well-known attacks fail in breaking the given protocol. In the second, a mathematically-convincing proof relates the computational difficulty of breaking the protocol to the difficulty of solving a presumed to be infeasible task.

2 Cryptographic Primitives

The protocols we describe in the next sections basically answer the question of *how groups of users can establish secret keys for subsequent cryptographic uses.* However, they require some preliminary notions and familiarity with certain cryptographic primitives. To this aim, we briefly recall, in a very simple way, some notions and definitions. For a complete treatment the reader can consult [94] and [120]. We start by recalling what a cryptosystem is:

Definition 1. *[120] A cryptosystem is a five-tuple* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *where the following conditions are satisfied:*

1. $\mathcal{P}$ *is a finite set of possible plaintexts*
2. $\mathcal{C}$ *is a finite set of possible ciphertexts*
3. $\mathcal{K}$*, the keyspace, is a finite set of possible keys*
4. *For each* $K \in \mathcal{K}$ *there is an encryption rule* $e_K \in \mathcal{E}$ *and a corresponding decryption rule* $d_K \in \mathcal{D}$*. Each* $e_K : \mathcal{P} \to \mathcal{C}$ *and* $d_K : \mathcal{C} \to \mathcal{P}$ *are functions such that* $d_K(e_K(x)) = x$ *for every plaintext element* $x \in \mathcal{P}$*.*

In a *symmetric* cryptosystem the key is a single secret element K, used by both the encryption and the decryption rules. Vice versa, in a *public key* cryptosystem, the key $K = (p, s)$ is a pair of elements: the first one p, the *public key*, is publicly known and can be used by everybody to encrypt messages to the owner of the key. On the other hand, the second one s, the *private key*, is held and used only by the owner to decrypt the messages sent to him. The main property of a public key cryptosystem is that the knowledge of p does not enable to compute (in a feasible way) s. Hence, public key cryptosystems can *only* be computationally secure.

Definition 2. *[120] A hash family is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where the following conditions are satisfied*

1. *$\mathcal{X}$ is a set of possible messages*
2. *$\mathcal{Y}$ is a finite set of possible message digests or authentication tags*
3. *$\mathcal{K}$, the key space, is a finite set of possible keys*
4. *For each $K \in \mathcal{K}$, there is a hash function $h_K \in \mathcal{H}$. Each $h_K : \mathcal{X} \to \mathcal{Y}$.*

Hash functions are used to associate a message digest to a certain message of arbitrary size, for example a file of data. The message digest can be used later on to check if the file has been corrupted. Some hash functions do not require keys (i.e. unkeyed hash functions). The main security property that hash functions satisfy is that it is computationally infeasible to find two messages which the hash function associates to the same message digest. This property, called *collusion resistance* implies that the function is *one-way*: in other words, it cannot be inverted by means of feasible computations.

Definition 3. *[120] A signature scheme is a five-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ where the following conditions are satisfied:*

1. *$\mathcal{P}$ is a finite set of possible messages*
2. *$\mathcal{A}$ is a finite set of possible signatures*
3. *$\mathcal{K}$, the keyspace, is a finite set of possible keys*
4. *For each $K \in \mathcal{K}$, there is a signing algorithm $sig_K \in S$ and a corresponding verification algorithm $ver_K \in V$. Each $sig_K : \mathcal{P} \to \mathcal{A}$ and $ver_K : \mathcal{P} \times \mathcal{A} \to \{true, false\}$ are functions such that the following equation is satisfied for every message $x \in \mathcal{P}$ and for every signature $y \in \mathcal{A}$:*

$$ver(x, y) = \begin{cases} true \text{ if } y = sig(x) \\ false \text{ if } y \neq sig(x). \end{cases}$$

A pair (x, y) with $x \in \mathcal{P}$ and $y \in A$ is called a signed message.

A signature scheme enables a user to sign messages. A signature is a short sequence of bits that *only* the owner of the message can produce. Everybody else can verify the authenticity of the signature on the message.

Notice that the use of public key cryptosystems implicitly assumes that a certain public key *really* corresponds to a given user. In other words, the identity of each user is *binded* to the key. This *authentication process* for the public keys can be done by using a trusted third party $\mathcal{TA}$ and a signature scheme. If the verification algorithm of the signature scheme held by $\mathcal{TA}$ is universally known and recognized to belong to the $\mathcal{TA}$, then the $\mathcal{TA}$ can fill in and sign a *certificate* for each public key, containing several information (i.e., public key, identity of the user, date of issue, expiring date ...). Then, every user can show the certificate to prove the authenticity of his own public key. The certificate can be verified by any other user of the system.

Most of the schemes we consider are designed over finite groups.

Definition 4. *Let G be a finite set of elements, and let $*$ be an operator defined on G. The pair $(G, *)$ is a* group *if*

- *G is closed with respect to $*$, i.e., $a * b \in G$ for any $a, b \in G$.*
- *$*$ is associative, i.e., $(a * b) * c = a * (b * c)$.*
- *1 is the identity element, i.e., $a * 1 = 1 * a = a$, for any $a \in G$.*
- *Any $a \in G$ has an* inverse *a^{-1} such that $a * a^{-1} = a^{-1} * a = 1$.*

The *order* of an element g of the group G is the smallest positive integer m such that $g^m = 1$, where g^m denotes the application of $*$ m times (i.e., $g^2 = g * g, g^3 = g * g * g$, etc...). An element g is a *primitive element* of the group G if $\{g^i : 0 \leq i \leq |G| - 1\} = G$. Denoting by $Z_p = \{0, \ldots, p-1\}$, where p is a large prime, the pair $(Z_p^*, \cdot)$, where $Z_p^* = Z_p \setminus \{0\}$, and $\cdot$ is the usual multiplication mod$p$ among numbers, is a widely used group.

3 Key Establishment

In this section we overview methods and ideas proposed during the last years to solve the key establishment problem. The two main approaches to key establishment developed in the literature are *Key Distribution* and *Key Agreement*. In the first case, as the words suggest, keys are given to the users towards a sort of distribution, often performed or helped by a trusted party. In the second, users are required to interact, by exchanging messages among each other, and to perform private computations, in order to agree on a common key. Varieties of protocols have been described, which can be classified according to the above criterion. Following the exposition given in [94], we start with some definitions.

Definition 5. *A* Key Establishment Protocol *provides a shared secret to two or more parties, for subsequent cryptographic use.*

The basic requirement that a key establishment protocol should satisfy is that any other party of the network should be unable to get the same key (or partial information about it), established by a given group. This roughly define a *secure* key establishment protocol.

Moreover, a very nice feature is that all the parties are aware of the identities of the other parties that can get the same secret key. More precisely, we can state the following:

Definition 6. *An* Authenticated Key Establishment Protocol *is a Key Establishment Protocol whereby the parties are assured of the identities of the other parties that may gain access to a particular secret key.*

Notice that an authenticated protocol just ensures *who are* the other parties that could get the key, but it *does not ensure* that they really hold the key. In other words, there is no confirmation that the key has really been computed by all the parties that are supposed to. Therefore, the authentication is a sort of *implicit* authentication.

Definition 7. *A* Key Confirmation Protocol *proves the real possession of a secret key held by a set of parties.*

If an authenticated key establishment protocol provides even key confirmation, the keys the parties get are said to be *explicitly* authenticated.

The protocols we present achieve some of the notions we have just given. We start by surveying methods based on public key cryptography and, hence, computationally secure. Then, we consider unconditionally secure key establishment protocols. The schemes given in the next subsection are all well described in textbook for Cryptography courses. We just recall them to point out the idea on which they are based on, but the reader is referred to [94,120] for proofs and details.

3.1 Computationally Secure Public-Key Based Schemes

Diffie and Hellman [54], in 1976, described a solution for the key establishment problem that enabled, for the first time, to avoid the preliminary meeting in a secret place. Their landmark paper, moreover, introduced the ideas of public key cryptosystem and digital signature scheme, even if the first real scheme was given in [107]. Recently, it has been pointed out that the same ideas were previously discovered by researchers at Bletchley Park [20], but were kept secret due to military reasons. The interested reader is referred to [111] for a detailed and pleasant historical reconstruction.

The scheme proposed by Diffie and Hellman is very simple and works as follows:

Diffie-Hellman Scheme
Let p be a large prime and let g be a generator of Z_p^*.

1. *Alice* chooses a random value $2 \leq x \leq p-2$ and sends g^x to *Bob*.
2. *Bob* chooses a random value $2 \leq y \leq p-2$ and sends g^y to *Alice*.
3. *Alice* and *Bob* compute the common key

$$g^{xy} = (g^x)^y = (g^y)^x.$$

The security of the scheme is based on the difficulty of computing the discrete log in Z_p^*. More precisely,

Definition 8. *Let p be a prime and let Z_p^* be the multiplicative over Z_p. Let g be a generator of Z_p^*. Given $a \in Z_p^*$, the value x such that $g^x = a$ is called the discrete log (or index) of a with respect to g.*

If p is a large prime, computing the discrete log in Z_p^* is presumed to be computationally infeasible. The best known algorithms at the state of the current knowledge require sub-exponential time in the size of p. In the literature, the computation of the discrete log is referred to as the *Discrete Log Problem*, (*DL*, for short).

The idea of the Diffie-Hellman scheme can be easily generalized to groups of more users. It is just necessary to exchange information in a circular way. For 3 users, for example, the scheme works as follows:

Generalised Diffie-Hellman Scheme
Let p be a large prime and let g be a generator of Z_p^*.

1. *Alice* chooses a random value $2 \leq x \leq p-2$ and sends g^x to *Bob*.
2. *Bob* chooses a random value $2 \leq y \leq p-2$ and sends g^x, g^y and g^{xy} to *Cher*.
3. *Cher* chooses a random value $2 \leq z \leq p-2$ and sends g^{yz} to *Alice* and g^{xz} to *Bob*.
4. *Alice*, *Bob* and *Cher* compute the common key

$$g^{xyz} = (g^{yz})^x = (g^{xz})^y = (g^{xy})^z.$$

One of the disadvantage of the above extension of the Diffie-Hellman scheme is that, when the number n of users grows up, the scheme requires $O(n)$ communication steps. The interested reader is referred to [117,118] for 'natural' extensions of the Diffie-Hellman key exchange. Moreover, he can consult some recent papers [37,38,39,40] and the references therein quoted.

Notice that, the Diffie-Hellman scheme (and its extensions) can be implemented in any group G, instead of Z_p^*, which is supposed to be difficult for the DL problem.

From a security point of view, these schemes are secure against an adversary, said to be *passive*, who just listen the conversation: indeed, due to the difficulty of the discrete log problem, the knowledge of g^x and g^y, does not enable to compute x and y and, hence g^{xy}. On the other hand, seems that there is no better way of using g^x and g^y to compute g^{xy}. The computation of g^{xy} given g^x and g^y is usually referred to as the *Diffie-Hellman* problem (*DH*, for short). There is no general reduction at the state of the current knowledge of the *DL* problem to the *DH* problem, even if in the last years it has been shown [93] that it is possible to construct groups for which breaking the Diffie-Hellman protocol is provably *as hard as* computing discrete logarithms and this equivalence holds for any group if a number theoretic conjecture holds[2].

Notice that the Diffie Hellman Scheme can be used in a *non-interactive* fashion if each user U_i publishes his choice/public-key $y_i = g^{a_i}$ and uses a_i to compute the common key shared with another user. More precisely, to compute the common key with user U_j he computes $(y_j)^{a_i} = (g^{a_j})^{a_i}$.

With this approach the key between any pair of users is fixed forever, while with the interactive version of the protocol, *freshness* of the key is guaranteed. In each session the users can compute a new key.

The Diffie-Hellman scheme can be subject to *active* attacks: an active adversary can modify or inject messages along the channel. A common strategy that can be applied is the so called *meet in the middle* attack. This strategy can be described as follows:

[2] The security of the DL and of the Diffie-Hellman problems has been studied in several papers. To name few, see [33,35,114].

Meet-in-the-Middle-Attack. Assume that *Eve* intercepts and changes the messages sent, according to the steps of the protocol, by *Alice* to *Bob* and vice versa. More precisely, *Eve* intercepts g^x and sends $g^{x'}$ to *Bob*. Then, *Eve* intercepts the reply g^y that *Bob* sends to *Alice*, computes and sends $g^{y'}$ to *Alice*. At this point *Eve* shares $g^{xy'}$ with *Alice* and $g^{x'y}$ with *Bob*. She can filter the conversation, while *Alice* and *Bob* think they are talking to each other.

Matsumoto, Takashima and Imai have constructed several interesting key agreement protocols by modifying the Diffie-Hellman protocol. The following MTI scheme [89] has been designed to cope with meet-in-the-middle attacks.

MTI Scheme
Let p be a large prime and let g be a generator of Z_p^*. Moreover, let $P_A = g^a$ be *Alice*'s public key and let $P_B = g^b$ be *Bob*'s public key. The public keys are certified by a trusted authority $\mathcal{TA}$.

1. *Alice* chooses a random value $2 \leq x \leq p-2$ and sends g^x to *Bob*.
2. *Bob* chooses a random value $2 \leq y \leq p-2$ and sends g^y to *Alice*.
3. *Alice* and *Bob* compute the common key

$$k = (g^y)^a P_B^x = (g^x)^b P_A^y = (g^{bx+ay}).$$

The use of the public keys mutually authenticate the users. In other words, both users are sure of the identity of the other party. However, the authentication is implicit since there is no key confirmation. In this scheme *Eve* can still avoid that *Alice* and *Bob* establish a common key but the meet-in-the-middle attack does not work.

Notice that, even in the non-interactive version of the DH protocol, if the public key $y_i = g^{a_i}$ is certified by a trusted authority, the key establishment scheme provides implicit authentication.

Another well-known variant of the Diffie-Hellman protocol is the Station-to-Station protocol (STS, for short). This scheme, introduced by Diffie, Van Oorschot, and Wiener [55], uses a symmetric cryptosystem and a digital signature scheme.

STS Scheme
Let p be a large prime and let g be a generator of Z_p^*. Moreover, let (P_A, S_A) be *Alice*'s public and private keys, and let (P_B, S_B) be *Bob*'s public and private keys. The public keys are certified by a trusted authority $\mathcal{TA}$. Finally, let E be a symmetric encryption scheme.

1. *Alice* chooses a random value $2 \leq x \leq p-2$ and sends g^x to *Bob*.
2. *Bob* chooses a random value $2 \leq y \leq p-2$ and sends g^y and $E_k(S_B(g^x, g^y))$ to *Alice*.
3. *Alice* sends to *Bob* $E_k(S_A(g^x, g^y))$.

The scheme provides explicit authentication. Key confirmation is given by means of the encryption E_k where $k = g^{xy}$.

Other interesting versions of the Diffie-Hellman scheme are represented by the so called *Gunther's Scheme* [69] and *Girault's Scheme* [66]. In these cases, the keys the user gets are *implicitly-certified* or *self-certified*. The scheme still requires a trusted authority. Moreover, in the following scheme, a hash function h is used by the parties.

Gunther's Scheme for implicitly-certified keys

1. The trusted authority $\mathcal{TA}$ selects a prime p and a generator g of Z_p^*. Moreover, $\mathcal{TA}$ selects a random $1 \leq t \leq p-2$ such that $\gcd(t, p-1) = 1$ as its private key, and publishes its public key $u = g^t \bmod p$, along with g and p.
2. $\mathcal{TA}$ assigns to each party A an identifier I_A and a random value k_A subject to $\gcd(k_A, p-1) = 1$. Then, $\mathcal{TA}$ computes $P_A = g^{k_A} \bmod p$ and solves for a the equation
$$h(I_A) = t \cdot P_A + k_A \cdot a \bmod (p-1).$$
3. $\mathcal{TA}$ securely sends to A the pair (P_A, a).
4. Any other party can reconstruct A's public key $(P_A)^a$ by computing
$$P_A^a = g^{h(I_A)} \cdot u^{-P_A} \bmod p.$$

The aim of this procedure is to avoid the *overhead* due to the use of certificates. Indeed, in this case there is no certificate associate with the keys but every user is guaranteed that P_A^a belongs to A, due to the procedure applied by $\mathcal{TA}$ to generate the public keys. Implicitly-certified keys can be used to set up variants of the DH protocols. For example:

Gunther's Key Agreement Scheme

1. *Alice* sends (I_A, P_A) to *Bob*.
2. *Bob* chooses a random value y, and sends $(I_B, P_B, P_A^y \bmod p)$ to *Alice*.
3. *Alice* sends to *Bob* $(P_B)^x \bmod p$.
4. *Alice* and *Bob* compute the same key k as
$$k = (P_A^y)^a (P_B^b)^x = (P_A^a)^y (P_B^x)^b = g^{k_A y a + k_B b x}.$$

The reader is referred to [66] for the Girault's scheme, where the key are *self-certifying*, i.e.,only the user knows the corresponding private key, compared to the Gunther's scheme. More details and references can be found in [94,120].

3.2 Key Transport

All the protocols described before enable two or more parties to agree on a common secret key. Each party *plays a role* in establishing the key. In this section we describe a smart technique, attributed to Shamir [80], enabling one party to *send* to another party a secret key for subsequent cryptographic uses.

Shamir's idea is the following: *Alice* chooses a key K, puts it in a box with a lock, and sends the box to *Bob*. *Bob* adds another lock and sends it back to *Alice*. *Alice* removes her lock and sends again the box to *Bob*. At this point, *Bob* removes his lock, opens the box, and recovers the key K. Therefore, with a 3-step protocol, they obtain a common key (chosen by *Alice*).

Shamir's Scheme
Let p be a prime and let Z_p^* be the multiplicative group over Z_p.

1. *Alice* and *Bob* choose secret random numbers a and b, coprime with $p-1$, and compute a^{-1} and b^{-1}, respectively.
2. *Alice* chooses a key K and sends $K^a \bmod p$ to *Bob*.
3. *Bob* computes and sends $(K^a)^b \bmod p$ to *Alice*
4. *Alice* computes and sends $(K^b) = (K^{ab})^{a^{-1}} \bmod p$ to *Bob*.

At the end of the execution both share the key K. The protocol is based on the DL problem but it can be rewritten using any suitable symmetric encryption scheme. However, some attention is required since, for example, if one uses the Vernam cipher, then the xor of the three messages exchanged gives the key K!

Notice that Shamir's scheme enables one party *to transport* a key to another, assuming that the two parties do not share a priori a secret key. Instead, assuming that both users already share a long term key, several techniques to establish a session key have been proposed, from very simple ones, where one party encrypts and sends the key to the other party, to more refined *challenge-response* protocols [94]. As we will point out later, session keys are useful for many reasons and in several settings. To exemplify the approach, we describe a protocol which provides mutual entity authentication (i.e., each entity is guaranteed of the identity and availability of the other) and implicit key authentication, and is based on symmetric primitives.

In the following scheme [10], we assume that *Alice* and *Bob* share two long-term symmetric keys K and K'. Moreover, h_K is a keyed hash function, used for entity authentication, and $h_{K'}$ is a keyed hash function, used to compute the session key.

Authenticated Key Exchange Protocol (AKEP2)
Let id_A and id_B be *Alice*'s and *Bob*'s identifiers.

1. *Alice* generates and sends a random number r_A to *Bob*.
2. *Bob* replies with the message $(T, h_K(T))$,
   where $T = (id_B, id_A, r_A, r_B)$ and r_B is a random number.
3. *Alice* sends $(id_A, r_B), h_K(id_A, r_B)$.
4. *Alice* and *Bob* compute the session key as $S = h_{K'}(r_B)$

The interpretation of the steps is quite straightforward. Key authentication is implicit since there is no confirmation at the end of the protocol. Entity Authentication is obtained by using h_K and the random numbers r_A, r_B.

Session keys can even be established by using public key techniques which go from the trivial solution of one party that generates and sends the session key to the other, to complex and well-designed schemes which use public key cryptosystems and digital signature schemes. To give an example of this approach, we describe one protocol of the standard $X.509$ [75]. It provides mutual entity authentication and implicit key authentication.

X.509 *Strong Two-way Authentication (Simplified Version)*

1. *Alice* constructs a message $M_A = (t_A, r_A, B, P_B(k_1))$ and sends to *Bob*
$$cert_A, M_A, Sign_A(M_A).$$
2. *Bob* constructs a similar message $M_B = (t_B, r_B, A, r_A, P_A(k_2))$ and sends to *Alice*
$$cert_B, M_B, Sign_B(M_B).$$

The protocol requires two steps. The messages M_A and M_B contain time stamps t_A, t_B, random numbers r_A, r_B, public identifiers A and B of *Alice* and *Bob*, and the encryptions with public keys of the secret values k_1, k_2, chosen by *Alice* and *Bob*, respectively. Each user sends to the other the message, his own signature of it, and a certificate for his/her public key. At the end of the protocol they share two secrets, implicitly authenticated. Time stamps and random numbers are used to avoid attacks, called *reply* attacks, in which the adversary stores and re-sends later on the same message, in order to share a key with one of the parties.

Many other protocols, based on the use of the same cryptographic primitives, providing slightly different messages and number of steps, have been proposed in the recent years. Some interesting protocols which use both symmetric primitives and public key primitives to establish session key, have been described as well. The Beller-Yacobi [14,15] is a well-known example of these schemes, which are said to be *hybrid* schemes.

To close this brief overview of computationally secure key establishment schemes, we would like to stress one more time the existence of a large number of papers that concern with this topic. The literature is really rich. And we would like just to give to the interested reader some more references about papers that he can decide to consult, like [2,4,8,10,12,13,16,19,31,36,41,42,45,46,50,51,52,53], [55,56,60,67,70,82,79,83,86,102,110,115,125,126,127,129,130,131,132]. Such a list is absolutely not exhaustive of the work that has been done in the last years, as the reader can find out browsing journals and conference proceedings related to cryptography and theoretical computer science in general.

3.3 Unconditionally Secure Schemes

Key establishment protocols secure against an unbounded adversary are said to be unconditionally secure: in other words, their security is not related to computational assumptions on the power of the adversary and on the amount of resources he can have access to. In this setting, the properties the protocol must satisfy are given by using the tools of the Probability Theory. Further, several definitions can be easily stated by using Information Theory and the Entropy Function. Since in our presentation we are going to use such tools, we start by briefly recalling some notions. Most of the material of these subsections can be found in [121], which is a complete overview of unconditionally secure key predistribution schemes and broadcast encryption schemes.

Information Theory Background. Let $\mathbf{X}$ be a random variable taking values on a set X according to a probability distribution $\{P_{\mathbf{X}}(x)\}_{x\in X}$. The *entropy* of $\mathbf{X}$, denoted by $H(\mathbf{X})$, is defined as

$$H(\mathbf{X}) = -\sum_{x\epsilon X} P_{\mathbf{X}}(x)\log P_{\mathbf{X}}(x),$$

where the logarithm is relative to the base 2. The entropy satisfies

$$0 \leq H(\mathbf{X}) \leq \log|X|,$$

where $H(\mathbf{X}) = 0$ if and only if there exists $x_0 \in X$ such that $Pr(\mathbf{X} = x_0) = 1$; whereas, $H(\mathbf{X}) = \log|X|$ if and only if $Pr(\mathbf{X} = x) = 1/|X|$, for all $x \in X$. The entropy of a random variable is usually interpreted as a measure of the:

- *"Equidistribution" of the random variable.* In this case, the entropy function is simply a mathematical function which says if the distribution of the random variable is close (i.e., $H(\mathbf{X}) \approx \log|X|$) or far (i.e., $H(\mathbf{X}) \approx 0$) from the uniform one.
- *Amount of information given on average by the random variable.* Assume that the random variable represents an experiment, and we have to take a decision depending on its outcome. Then, if the result is determined (i.e., $H(\mathbf{X}) = 0$), it gives us no information in order to take the decision. We can decide without looking at the experiment because we already know what will be the result. On the other hand, if the output is totally random (i.e., $H(\mathbf{X}) = \log|X|$), the knowledge of the result can help us (i.e., gives information) about the appropriate decision.

Given two random variables $\mathbf{X}$ and $\mathbf{Y}$, taking values on sets X and Y, respectively, according to a probability distribution $\{P_{\mathbf{XY}}(x,y)\}_{x\in X, y\in Y}$ on their Cartesian product, the *conditional entropy* $H(\mathbf{X}|\mathbf{Y})$ is defined as

$$H(\mathbf{X}|\mathbf{Y}) = -\sum_{y\in Y}\sum_{x\in X} P_{\mathbf{Y}}(y)P_{\mathbf{X}|\mathbf{Y}}(x|y)\log P_{\mathbf{X}|\mathbf{Y}}(x|y).$$

Since $H(\mathbf{X}|\mathbf{Y})$ can be re-written as $\sum_{y \in Y} P_{\mathbf{Y}}(y) H(\mathbf{X}|\mathbf{Y} = y)$, it follows that

$$H(\mathbf{X}|\mathbf{Y}) \geq 0. \tag{1}$$

with equality if and only if $\mathbf{X}$ is a function of $\mathbf{Y}$. Along the same line, the conditional entropy is a measure of the amount of information that $\mathbf{X}$ "still has", once given $\mathbf{Y}$.

The *mutual information* between $\mathbf{X}$ and $\mathbf{Y}$ is given by

$$I(\mathbf{X}; \mathbf{Y}) = H(\mathbf{X}) - H(\mathbf{X}|\mathbf{Y}).$$

Since, $I(\mathbf{X}; \mathbf{Y}) = I(\mathbf{Y}; \mathbf{X})$ and $I(\mathbf{X}; \mathbf{Y}) \geq 0$, it is easy to see that

$$H(\mathbf{X}) \geq H(\mathbf{X}|\mathbf{Y}), \tag{2}$$

with equality if and only if $\mathbf{X}$ and $\mathbf{Y}$ are independent. The mutual information is a measure of the common information between $\mathbf{X}$ and $\mathbf{Y}$.

The protocols we discuss later on can be concisely described by using a common framework. *Key Predistribution Schemes, Key Agreement Schemes* and *Broadcast Encryption Schemes*, can all be defined in terms of the entropy function by means of few equations. Thus, we start by outlining the model we consider in the following:

Model. Let $\mathcal{TA}$ be a trusted authority and let $\mathcal{U} = \{1, \ldots, n\}$ be a set of users. Each user is connected with the $\mathcal{TA}$ by means of a *private channel*. Moreover, $\mathcal{TA}$ and users have access to a *broadcast channel*.

In a Key Predistribution Scheme the $\mathcal{TA}$ generates and distributes secret information to each user along the private channels. The secret information enables later on several subsets of users to compute secret keys. More precisely, if $2^{\mathcal{U}}$ denotes the set of all subsets of users $\mathcal{U}$, we define $\mathcal{P} \subseteq 2^{\mathcal{U}}$ to be the family of *privileged subsets* of $\mathcal{U}$ who need a common key, and $\mathcal{F} \subseteq 2^{\mathcal{U}}$ to be the family of *forbidden subsets*, i.e., the possible coalitions of $\mathcal{U}$ against whom each key must to remain secure. After the *distribution phase* performed by the $\mathcal{TA}$, each privileged subset $P \in \mathcal{P}$ is able to compute the key k_P associated with P. On the other hand, no forbidden subset $F \in \mathcal{F}$, disjoint from P, is able to compute any information about k_P. We stress that in such schemes each user computes the keys by using the secret information and possible some public information available across the system, but no interaction either with the users or with the $\mathcal{TA}$ is required. In a certain way, the keys are predetermined by the secret information.

The information given to user i through a private channel can be denoted, for $i = 1, \ldots, n$, by $u_i \in U_i$, where U_i represents a set of possible values. Moreover, for any subset $X = \{i_1, \ldots, i_k\} \subseteq \mathcal{U}$, we denote by $U_X = U_{i_1}, \ldots, U_{i_k}$ the usual Cartesian product.

We assume that there is a probability distribution on $U_{\mathcal{U}}$, and the $\mathcal{TA}$ chooses $u_{\mathcal{U}} \in U_{\mathcal{U}}$ according to this probability distribution. Using the above notation, we can state the following:

Definition 9. *A $(\mathcal{P}, \mathcal{F})$-Key Predistribution Scheme ($(\mathcal{P}, \mathcal{F})$-KPS, for short) is a protocol divided in two phases: a* distribution phase*, performed by the $\mathcal{TA}$, and a* key computation phase*, performed by the users, satisfying the following properties:*

- *Each user i in any privileged set P can compute k_P. More formally, for all $i \in P$,*
$$H(\mathbf{K}_P|\mathbf{U}_i) = 0.$$
- *No forbidden subset F, disjoint from any privileged subset P, has any information on k_P. More formally, for all $P \in \mathcal{P}$ and $F \in \mathcal{F}$ such that $P \cap F = \emptyset$,*
$$H(\mathbf{K}_P) = H(\mathbf{K}_P|\mathbf{U}_F).$$

A trivial Key Predistribution Scheme consists in giving to each possible subset P of privileged users a secret key κ_P.

Basic KPS
- *Distribution Phase.* The $\mathcal{TA}$ chooses a value $k_P \in \mathcal{K}$ for each $P \in \mathcal{P}$ and gives the value to every user $i \in P$.
- *Key Computation Phase.* Every user i just looks up in his or her memory the key k_P.

Notice that with this solution there is no real key computation phase: each user gets the keys corresponding to the groups in which he belongs. Moreover, it is easy to see that any coalition $F \cap P = \emptyset$ has no information on k_P.

The main problem with the above scheme is the *large amount of secret keys* that each user has to store. Using the language of Information Theory, we can say that the efficiency of a KPS is measured by the amount of secret information that the $\mathcal{TA}$ distributes and that each user has to share. More precisely, two measures, the *information rate* and the *total information rate*, are defined respectively as

$$\rho = \min_{i=1,\ldots n} \frac{H(\mathcal{K})}{H(\mathbf{U}_i)} \quad \text{and} \quad \rho_T = \frac{H(\mathcal{K})}{H(\mathbf{U}_\mathcal{U})}.$$

The first measure is the minimum ratio between the size of the secret key and the size of the secret information given to the user. The second is the ratio between the size of the secret key and the size of the total secret information given to the users in $\mathcal{U}$.

Coming back to the *Basic Scheme*, if $\mathcal{P}$ is the set of all subsets of $\mathcal{U}$ of size t we can denote the $(\mathcal{P}, \mathcal{F})$-KPS as a $(t, \mathcal{F})$-KPS. Along the same line, if $\mathcal{P}$ is the set of all subsets of $\mathcal{U}$ of size at most t we will use the notation $(\leq t, \mathcal{F})$-KPS. Moreover, if $\mathcal{F}$ is the set of all subsets of $\mathcal{U}$ of size (at most) ω, we will refer to a $(\mathcal{P}, \omega)$-KPS, ($(\mathcal{P}, \leq \omega)$-KPS, respectively).

From the above construction, easily follows the next results:

Theorem 1. *For any $t > 1$, there is a $(t, \leq n)$-KPS having information rate and total information rate equal to*

$$\rho = \frac{1}{\binom{n-1}{t-1}} \qquad \text{and} \qquad \rho_T = \frac{1}{\binom{n}{t}}.$$

If $t = 2$, the above result states that the basic scheme enables any pair of users to privately communicate against any disjoint coalition of at most $n-2$ users by given $n-1$ secret keys to each user. Further, the $\mathcal{TA}$ has to generate $\binom{n}{2} = \frac{n(n-1)}{2}$ keys. In the literature this large amount of keys that must be generated is well-known as the n^2 problem and was the motivation for further researches. Indeed, given the high complexity of such a distribution mechanism, a natural step is to trade complexity for security. We may still require that keys are unconditionally secure, but only with respect to coalitions of *a limited size*.

In order to reduce the number of keys that each user has to store and the $\mathcal{TA}$ has to generate in the *Basic Scheme*, Blom [21] introduced a scheme enabling a tradeoff between the number of keys that the user has to store and the size of a coalition of adversary that can break the scheme. The protocol he gave in [21] can be described as follows:

Blom's Scheme

- *Distribution Phase.* Let $q \geq n$. The $\mathcal{TA}$ chooses n distinct random numbers $s_i \in GF(q)$, and gives s_i to user i, for $i = 1, \ldots, n$. These values are public identifiers for the users. Then, the $\mathcal{TA}$ constructs a random bivariate polynomial

$$f(x, y) = \sum_{i=0}^{\omega} \sum_{j=0}^{\omega} a_{ij} x^i y^j,$$

  having coefficients in $GF(q)$, such that $a_{ij} = a_{ji}$ for all i, j.
- For $i = 1, \ldots, n$, the $\mathcal{TA}$ computes the polynomial

$$g_i(x) = f(x, s_i) = \sum_{j=0}^{\omega} b_{ij} x^j,$$

  and gives the $\omega + 1$ values b_{ij} to user i.
- *Key Computation Phase.* Users i and j compute the key

$$k_P = g_i(s_j) = g_j(s_i).$$

The original formulation of the scheme uses MDS codes [21], and the interested reader can consult [94] for the original description of Blom's scheme and some background on MDS code as well. Blom's scheme was reformulated in terms of symmetric polynomials in [26], where a generalization to the case of $(t, \leq \omega)$-KPS was given. More precisely:

Blundo's et al. Scheme

- *Distribution Phase.* Let $q \geq n$. The $\mathcal{TA}$ chooses n distinct random numbers $s_i \in GF(q)$, and gives s_i to user i, for $i = 1, \ldots, n$. These values are public identifiers for the users. Then, the $\mathcal{TA}$ constructs a random n-variate polynomial

$$f(x_1, \ldots, x_t) = \sum_{i_1=0}^{\omega} \cdots \sum_{i_t=0}^{\omega} a_{i_1 \ldots i_t} x^{i_1} \ldots x^{i_t},$$

  having coefficients in $GF(q)$, such that $a_{i_1 \ldots i_t} = a_{j_1 \ldots j_t}$ for any permutation $j_1 \ldots j_t$ of the set of indices $i_1 \ldots i_t$.
- For $i = 1, \ldots, n$, the $\mathcal{TA}$ computes and sends to user i the polynomial

$$g_i(x_2, \ldots, x_n) = f(s_i, x_2, \ldots, x_n)$$

- *Key Computation Phase.* Any set of t users $P = \{i_1, \ldots, i_t\}$ computes the key

$$k_P = g_{i_1}(s_{i_2}, \ldots, s_{i_t}) = \cdots = g_{i_t}(s_{i_1}, \ldots, s_{i_{t-1}}).$$

Blom's Scheme and its generalization, by a simple counting argument, lead to the following result:

Theorem 2. *For any $t \geq 2$ and $\omega \geq 1$, there exist a $(t, \leq \omega)$-KPS having information rate and total information rate equal to*

$$\rho = \frac{1}{\binom{t+\omega-1}{t-1}} \qquad \textit{and} \qquad \rho_T = \frac{1}{\binom{t+\omega}{t}}.$$

Moreover, in [26] it was shown, using Information Theory arguments, that the *Basic Scheme*, the *Blom's Scheme* and the *Blundo's et al. Scheme* are optimal in terms of information rate and total information rate.

Another $(\mathcal{P}, \mathcal{F})$-KPS was proposed by Fiat and Naor in [58]. It was presented as a zero-message broadcast encryption scheme (which will be defined later) but, as pointed out by Stinson, it turns out to be actually a KPS. More precisely, the scheme they described is an $(\leq n, \leq \omega)$-KPS.

Fiat-Naor Scheme

- For every subset $F \subseteq \mathcal{F}$, where $\mathcal{F}$ is the set of all subsets of cardinality at most ω, the $\mathcal{TA}$ chooses a random value $s_F \in GF(q)$ and sends s_F to every member of $\mathcal{U} \setminus F$.
- A priviliged subset P computes

$$k_P = \sum_{F \in \mathcal{F}: F \cap P = \emptyset} s_F.$$

It is easy to see that a key k_P, computed by the set of users P, is secure against any $F \in \mathcal{F} : F \cap P = \emptyset$ since no user belonging to the subset F gets the value s_F associated with F.

Stinson, in his survey [121], pointed out that the *Basic Scheme* and the *Fiat-Naor Scheme* can be seen as instances of a more general construction based on the idea of key distribution patterns, introduced by Mitchell and Piper in [95]. (For constructions see also [105,122]).

Definition 10. *Let $\mathcal{B} = \{B_1, \ldots, B_\beta\}$ be a set of subsets of $\mathcal{U}$. The pair $(\mathcal{U}, \mathcal{B})$ is a $(\mathcal{P}, \mathcal{F})$-Key Distribution Pattern ($(\mathcal{P}, \mathcal{F})$-KDP for short) if for all $P \in \mathcal{P}$ and $F \in \mathcal{F}$ such that $P \cap F = \emptyset$ it results:*

$$\{B_j : P \subseteq B_j \text{ and } F \cap B_j = \emptyset\} \neq \emptyset.$$

Loosely speaking, the above definition requires that each $P \in \mathcal{P}$ is "embedded" in a $B_j \in \mathcal{B}$, disjoint from all $F : F \cap P = \emptyset$.

A $(\mathcal{P}, \mathcal{F})$-KPS scheme can be constructed by using a $(\mathcal{U}, \mathcal{B})$-KDP as follows:

KDP-Based Scheme

- For every subset $B_j \in \mathcal{B}$ the $\mathcal{TA}$ chooses a random value $s_{B_j} \in GF(q)$ and sends s_{B_j} to every user in B_j.
- A priviliged subset P computes

$$k_P = \sum_{B_j : P \subseteq B_j} s_{B_j}.$$

The scheme works because every user $i \in P$ can compute the key, i.e., if $i \in P$ then $i \in B_j$. Hence, he gets s_{B_j} for all $B_j : P \in B_j$. On the other hand, every F will miss at least one value s_{B_j} for a subset B_j such that $P \subseteq B_j$ and $B_j \cap F = \emptyset$.

Many examples of such a construction are given in [121], and the interested reader is strongly encouraged to read that paper.

The main drawback of Key Predistribution Schemes lies in the high memory storage requirement. In order to avoid such heavy requirement, a second approach to the key establishment problem, allowing interaction among the users to compute a common key, was introduced. More precisely, during the Key Computation Phase, the members of a group G, using the secret information received in the Distribution Phase, interact to agree on a key, by exchanging encrypted messages among themselves via the broadcast channel. Any disjoint coalition of adversaries F that hears the communication is still unable to gain any information about it. This approach, usually referred to as *unconditionally secure key agreement*, initiated in [26], was continued by Beimel and Chor [6,7] and it was aimed to reduce the size of information each user *must keep secret*.

Denoting by $\mathbf{C}_i$ the random variable taking values on the set C_i and representing the messages received by user U_i during the key computation phase, sent by the other users of the system, and using again the language of Information Theory, such schemes can be defined as follows:

Definition 11. *A $(\mathcal{P},\mathcal{F})$-Key Agreement Scheme ($(\mathcal{P},\mathcal{F})$-KAS, for short) is a protocol divided in two phases: a* distribution phase*, performed by the $\mathcal{TA}$, and a* key computation phase*, performed by the users, satisfying the following properties:*

- *Each user i in any privileged set P can compute k_P by using the private information received in the distribution phase and the messages received during the key computation phase. More formally, for all $i \in P$,*

$$H(\mathbf{K}_P|\mathbf{U}_i\mathbf{C}_i) = 0.$$

- *No forbidden subset F, disjoint from any privileged subset P, has any information on k_P. More formally, for all $P \in \mathcal{P}$ and $F \in \mathcal{F}$ such that $P \cap F = \emptyset$,*

$$H(\mathbf{K}_P) = H(\mathbf{K}_P|\mathbf{U}_F\mathbf{C}_F).$$

Even for key agreement schemes, the performances are measured by an *information rates*, a *communication rate* and a *total information rate*, defined respectively as

$$\rho = \min_{i=1,\ldots n} \frac{H(\mathcal{K})}{H(\mathbf{U}_i)}, \quad \rho_C = \min_{P\in\mathcal{P}} \frac{H(\mathcal{K})}{H(\mathbf{C}_P)}, \quad \text{and} \quad \rho_T = \min_{P\in\mathcal{P}} \frac{H(\mathcal{K})}{H(\mathbf{U}_{\mathcal{U}}\mathbf{C}_P)}.$$

The first measure is the minimum ratio between the size of the secret key and the size of the secret information given to the user. The second is the minimum ratio between the size of the secret key and the size of the messages received by users $i \in P$; while the third measure is the minimum ratio between the size of the secret key and the total secret information given to the users in $\mathcal{U}$ along with the messages exchanged $\mathbf{C}_P$ to compute a common key.

Unfortunately, in [6], the authors studied key agreement schemes for groups of users G of size g and coalitions of adversaries F of size b, and they proved that the interaction *cannot help* in reducing the size of the pieces of information given to the users compared to the non interactive model we have seen before. Hence, in order to decrease the size of the secret information, we have to relax the *security requirements*. We can require the key agreement scheme to be secure only *a fixed number* of times, say τ, defining *τ-restricted* key agreement schemes. In such schemes we limit to τ the number of groups of users, whose identity is not known beforehand, that can compute a common key in an unconditionally secure way. For such schemes Beimel and Chor in [6,7] realized a one-restricted scheme, where the size of pieces given to users is smaller than in unrestricted key agreement schemes. In the literature a one-restricted scheme is also referred to as a *one-time* scheme, because it can be used to compute only *one* common key by a single group of users of the system.

In [29] the authors presented a generalization of the one-restricted scheme proposed by Beimel and Chor [6,7] using tools from design theory. In order to give an example of an unconditionally secure Key Agreement Scheme, we describe this scheme [29]. However, we need some definitions and results from

design theory. Compared to other protocols we have seen before, the description of the following one is a bit more complicated but, at the same time, it is a good example of the elegant and refined use of combinatorial structures that often is done in Cryptography.

Definition 12. *A design is a pair* $(V, \mathcal{B})$*, where* V *is a set of* n *elements (called* points*) and* $\mathcal{B}$ *is a set of subsets of* V *of a fixed size* k*, where* $k \geq 2$*, (called* blocks*).*

Designs with suitable features are *resolvable* design.

Definition 13. *A* parallel class *of* $(V, \mathcal{B})$ *consists of* n/k *blocks from* $\mathcal{B}$ *which partition the set* V*. The design* $(V, \mathcal{B})$ *is said to resolvable if the set of blocks,* $\mathcal{B}$*, can be partitioned into parallel classes. If* $\mathcal{B}$ *consists of all* k*-subsets of* V*, then* $(V, \mathcal{B})$ *is called the complete* k*-uniform hypergraph on* V*.*

We will use the following theorem of Baranyai, a proof of which can be found in [84] (Theorem 36.1)

Theorem 3. *The complete* k*-uniform hypergraph on* n *points is resolvable if* $n \equiv 0 \bmod k$*.*

Notice that in the following the sets elements are being listed sequentially in increasing order.

A Protocol for one-restricted key agreement scheme: Let $\mathcal{U} = \{1, \ldots, n\}$ be a set of n users and let $G \subseteq \mathcal{U}$ be a group of users of size g. Suppose that $\ell \geq 2$ is an integer such that $g \equiv 1 \bmod (\ell - 1)$ and that $k \geq 1$ is an integer. The set-up phase consists of the $\mathcal{TA}$ distributing secret information corresponding to a Blundo's et al. $(\ell, b+g-\ell)$-KPS described before, implemented over $(Z_{p^k})^{\ell}$, with p prime. For an ℓ-subset of users A, we denote by k_A the key associated with A. We will think of k_A as being made up of ℓ independent keys over Z_{p^k}, which we denote by $k_{A,1}, \ldots, k_{A,\ell}$.

Each user h of a group G performs the following steps:

1. Chooses a random value $m^{(h)} = (m_1^h, \ldots, m_r^h) \in (Z_{p^k})^r$, where $r = \binom{g-2}{\ell-2}$.
2. Partitions the complete $(\ell-1)$-uniform hypergraph on $G \setminus \{h\}$ into r parallel classes $C_1, \ldots, C_r$, which all consist of $\chi = (g-1)/(\ell-1)$ blocks that we denote with $B_{i,j}^h$, for $1 \leq i \leq r$ and $1 \leq j \leq \chi$.
3. For each block $B_{i,j}^h$ denote with $B(i,j,h)$ the set $B_{i,j}^h \cup \{h\} = \{x_1, \ldots, x_\ell\}$, and let $\alpha_{i,j}^h$ denote the index such that $x_{\alpha_{i,j}^h} = h$.
4. Encrypts each m_i^h using the χ keys $k_{B(i,j,h),\alpha_{i,j}^h}$ by defining

$$b_{i,j}^h = k_{B(i,j,h),\alpha_{i,j}^h} + m_i^h \bmod p^k,$$

   for $1 \leq i \leq r$ and $1 \leq j \leq \chi$.
5. Broadcasts the vector

$$b^{(h)} = (b_{1,1}^h, \ldots, b_{1,\chi}^h \ldots, b_{r,1}^h, \ldots, b_{r,\chi}^h).$$

The secret key is the value $k_G = (m^{(1)}, \ldots, m^{(g)})$ which can be decrypted by anyone in G from the global broadcast $b_G = (b^{(1)}, \ldots, b^{(g)})$.

The next simple example illustrates the steps of this protocol.

Example 1. Suppose that $g = 5$ and $\ell = 3$. Note that $5 \equiv 1 \bmod 2$. Suppose that the group set is $G = \{1, 2, 3, 4, 5\}$. For each user $i \in G$, we partition the 2-subsets of $G \backslash \{i\}$ into $r = 3$ disjoint parallel classes. Below, we describe only the ones related to user 4.

$$C_1^4 = \{\{1, 2\}, \{3, 5\}\}, \qquad C_2^4 = \{\{1, 3\}, \{2, 5\}\},$$

$$C_3^4 = \{\{1, 5\}, \{2, 3\}\}.$$

Consider the computations performed by user 4. First, user 4 picks three random values (i.e., his part of the key), say $m_1^4, m_2^4, m_3^4 \in \mathbf{Z}_{p^k}$. Next, he computes the relevant α values. These are as follows:

$$\alpha_{1,1}^4 = 3,\ \alpha_{1,2}^4 = 2,\ \alpha_{2,1}^4 = 3,$$
$$\alpha_{2,2}^4 = 2,\ \alpha_{3,1}^4 = 2,\ \alpha_{3,2}^4 = 3.$$

This determines the values broadcasted by user 4:

$$b^{(4)} = (m_1^4 + k_{\{1,2,4\},3}, m_1^4 + k_{\{3,4,5\},2}, m_2^4 + k_{\{1,3,4\},3},$$
$$m_2^4 + k_{\{2,4,5\},2}, m_3^4 + k_{\{1,4,5\},2}, m_3^4 + k_{\{2,3,4\},3}).$$

△

The security of the above protocol derives from the observation that any coalition F of b users such that $F \cap G = \emptyset$, has no information about the key after the observation of the broadcast, even if they pool all their secret information. Indeed, as proved in Lemma 3.3 of [29], the $\binom{g}{\ell}$ keys used by the group appear to any disjoint coalition to be independent random elements of Z_{p^k}. Since each of these keys is used exactly once (the definition of the indices $\alpha_{i,j}^h$ ensures that every $k_{A,j}$ is used to encrypt exactly one $m_{i,j}$'s), they function as a series of one-time pads.

Notice that, using τ copies of a one-restricted scheme, we can set up a scheme which is secure for τ conferences. Such an approach, even though it allows us to construct a scheme in a straightforward manner, does not give rise to a scheme which is optimal with respect to the size of the information kept by each user [23].

The third approach to the Key Establishment Problem is represented by the so-called broadcast encryption schemes. In this case, the trusted authority $\mathcal{TA}$, during the distribution phase of the scheme distributes private information to the users, through the secure point-to-point channels. Later on, the $\mathcal{TA}$ enables a *privileged* subset P of the users to recover a common secret key by broadcasting an encrypted message, that only users in P can decrypt.

Denoting by $\mathbf{B}$ the random variable that takes values on the set B, representing the broadcast (encrypted) message sent by the $\mathcal{TA}$, a broadcast encryption scheme can be defined as follows:

Definition 14. *A $(\mathcal{P}, \mathcal{F})$-Broadcast Encryption Scheme ($(\mathcal{P}, \mathcal{F})$-BES, for short) is a protocol divided in two phases: a* distribution phase*, performed by the $\mathcal{TA}$, and a* key computation phase*, performed by the users and the $\mathcal{TA}$, satisfying the following properties:*

- *Each user i in a privileged set P can compute k_P by using the private information received in set up phase and the broadcast message sent by the $\mathcal{TA}$ during the key computation phase. More formally, for all $i \in P$,*

$$H(\mathbf{K}_P|\mathbf{U}_i\mathbf{B}_P) = 0.$$

- *No forbidden subset F, disjoint from any privileged subset P, has any information on k_P. More formally, for all $P \in \mathcal{P}$ and $F \in \mathcal{F}$ such that $P \cap F = \emptyset$,*

$$H(\mathbf{K}_P) = H(\mathbf{K}_P|\mathbf{U}_F\mathbf{B}_P).$$

For broadcast encryption schemes, the performances are measured by

$$\rho = \min_{i=1,\ldots n} \frac{H(\mathcal{K})}{H(\mathbf{U}_i)}, \quad \rho_B = \min_{P \in \mathcal{P}} \frac{H(\mathcal{K})}{H(\mathbf{B}_P)}, \quad \text{and} \quad \rho_T = \min_{P \in \mathcal{P}} \frac{H(\mathcal{K})}{H(\mathbf{U}_{\mathcal{U}}\mathbf{B}_P)}.$$

where the meaning is exactly the same holding for key agreement schemes, with the only difference that, instead of considering the messages exchanged, the above measures consider the *messages broadcasted* by the dealer during the broadcast phase.

The first broadcast encryption schemes we are going to consider are the one-level and multi-level schemes described in [58]. To get started, we recall the following definition:

Definition 15. *An (n, m, ω)-perfect hash family is a set $\mathcal{H}$ of functions*

$$f : \{1, \ldots, n\} \to \{1, \ldots, m\}$$

such that, for every subset $X \subset \{1, \ldots, n\}$ of size ω, there exists a function $f \in \mathcal{H}$ whose restriction f_X to X is one-to-one.

An (n, m, ω)-perfect hash family is usually denoted by $\mathrm{PHF}(N, n, m, \omega)$, where $|\mathcal{H}| = N$. Fiat and Naor, in their paper, gave some one-resilient schemes, i.e., schemes secure against attacks performed by one user. Then, by using a bunch of one-resilient BES schemes and a $\mathrm{PHF}(N, n, m, \omega)$, they set up an ω-resilient BES schemes.

A first (unconditionally secure) construction for one-resilient scheme is given by the so-called zero message broadcast encryption scheme that we have already presented in the context of key predistribution schemes (i.e., Fiat-Naor KPS). Moreover, two computationally secure one-resilient schemes were given. We describe the second one:

One-resilient BES based on a computational assumption

- The dealer chooses two large primes p, q and computes $n = pq$. It also chooses a *secret value* $g \in Z_n^*$. Then, for each user i, he computes and sends to the user a secret key $g_i = g^{p_i}$. The values $p_1, \ldots, p_n$ are public and such that, for each $i \neq j$, it results $p_i \neq p_j$.
- A privileged group G computes a common key g_G by using the public values $p_1, \ldots, p_n$. More precisely, user $i \in G$ can compute g_G by evaluating

$$g_i^{\prod_{j \in G \setminus \{i\}} p_j} \bmod n.$$

It is easy to see that each user in G computes the same key. Moreover, it is possible to show that if some user $j \notin G$ could compute the common key for G, then the user can even compute the secret value g chosen by the dealer. Therefore, assuming that *extracting roots modulo a composite n is hard*, the scheme is secure. For details the reader is referred to [58].

Using one-resilient schemes and a family of perfect hash functions, an ω-resilient scheme can be described as follows:

ω-resilient BES

For $1 \leq i \leq N$ and $1 \leq j \leq m$ let $R(i, j)$ be a $(n, 1)$-BES scheme, and let $\mathrm{PHF}(N, n, m, \omega)$ be a family of perfect hash functions.

- *Set up Phase.* The dealer sends to every user $i \in \{1, \ldots, n\}$ the keys associated with him by the scheme $R(i, f_j(i))$, for any $j = 1, \ldots, N$.
- *Broadcast Phase.* The dealer, to send message m, chooses $N - 1$ random elements $m_1, \ldots, m_{N-1}$ and computes

$$m_N = m_1 \otimes \cdots\cdots \otimes m_{N-1} \otimes m$$

- Then, he broadcasts, for $j = 1, \ldots, N$, the values m_j to the users belonging to $P \subset \{1, \ldots, n\}$ by means of the schemes $R(j, f_j(i))$, for any $i \in P$.

Every user in P can recover all the m_j's and can compute the message by a simple xor operation. On the other hand, the properties of the hash family guarantee that, for any subset $X = \{i_1, \ldots i_\omega\}$ of users, one of the function $f_j \in \mathcal{H}$ is one-to-one on X. Hence, the users in X cannot break any of the schemes $R(j, f_j(i_1)), \ldots, R(j, f_j(i_\omega))$ since they are one-resilient and can be broken only if at least two dishonest users are associated with the same scheme, i.e., $f_j(i_k) = f_j(i_\ell)$ for $k \neq \ell$. As a consequence, even if some user in P receives m_j by means of one of the schemes $R(j, f_j(i_1)), \ldots, R(j, f_j(i_\omega))$, the message m_j cannot be recovered by X. Therefore, m cannot be computed by X.

The above construction has been re-formulated by Stinson using some designs. The reader is referred to [121] for details. Notice that if the 1-resilient BES, used as a building block, is computationally secure, than the ω-resilient BES is computationally secure. On the other hand, an unconditionally secure 1-resilient BES implies an unconditionally secure ω-resilient BES.

A general construction for BES schemes has been proposed in [121,122]. The idea is to use basic Fiat-Naor Schemes in conjunction with an ideal secret sharing scheme (ISSS, for short). The goal in [122] was to obtain schemes where each user has to *store less values* and the broadcast messages are *shorter* compared to other constructions. In order to describe the construction we need to introduce before the concept of a secret sharing scheme.

Secret Sharing Schemes. A secret sharing scheme is a method by means of which a secret can be shared among a set $\mathcal{P}$ of n participants in such a way that qualified subsets of $\mathcal{P}$ can recover the secret, but forbidden subsets cannot. Secret sharing were introduced in 1979 by Blakley [5] and Shamir [112]. The reader can find an excellent introduction in [119]. The collection of subsets of participants qualified to reconstruct the secret is usually referred to as the *access structure* of the secret sharing scheme. Formally, we have:

Definition 16. *Let $\mathcal{P}$ be a set of participants, a* monotone *access structure Γ on $\mathcal{P}$ is a subset $\Gamma \subseteq 2^{\mathcal{P}} \setminus \{\emptyset\}$, such that*

$$A \in \Gamma, A \subseteq A' \subseteq \mathcal{P} \Rightarrow A' \in \Gamma.$$

A secret sharing scheme Σ is a protocol divided into two phases: a *distribution phase*, in which the dealer sends a secret piece of information, called *share*, to every participant, and a *reconstruction phase*, where the authorized subsets of participants, by pooling together their shares, reconstruct the secret. Any secret sharing scheme Σ for secrets in S and a probability distribution $\{p_S(s)\}_{s \in S}$ naturally induce a probability distribution on the secret information a held by the subset $A \subseteq \mathcal{P}$.

Denoting by $\mathbf{A}$ and $\mathbf{S}$ the random variables representing the possible shares received by $A \subset \mathcal{P}$ and the possible secret chosen by the dealer, in terms of Shannon's entropy we can state the following:

Definition 17. *A secret sharing scheme Σ is a* perfect *secret sharing scheme with secrets chosen in S, for the monotone access structure $\Gamma \subseteq 2^{\mathcal{P}}$ if*

1. Any subset of participants $A \in \Gamma$ can compute the secret: *Formally, for all $A \in \Gamma$, it holds that $H(\mathbf{S}|\mathbf{A}) = 0$.*
2. Any subset of participants $A \notin \Gamma$ has no information on the secret value: *Formally, for all $A \notin \Gamma$, it holds that $H(\mathbf{S}|\mathbf{A}) = H(\mathbf{S})$.*

Property 1 means that the value of the shares held by $A \in \Gamma$ completely determines the secret $s \in S$. On the other hand, Property 2 means that the probability that the secret is equal to s given that the shares held by $A \notin \Gamma$ are a, is the same as the *a priori* probability of the secret s.

The efficiency of a secret sharing scheme is measured by means of an "information rate", which relates the size of the secret with the size of the shares given to the participants. More precisely, given a secret sharing scheme Σ for the access structure Γ, on the set of secrets S, we define the information rate $\rho(\Sigma, \Gamma, S)$ as

$$\rho(\Sigma, \Gamma, S) = \frac{\log |S|}{\max_{P \in \mathcal{P}} \log |K(P)|},$$

where $K(P)$ is the set of possible share for participant P, and

$$\rho(\Gamma) = \sup \rho(\Sigma, \Gamma, S),$$

where the sup is taken over the space of all possible sets of secrets S, $|S| \geq 2$, and all secret sharing schemes for Γ. Secret sharing schemes with information rate equal to one, which is the maximum possible value of this parameter, are called *ideal*, and an access structure Γ on $\mathcal{S}$ is said to be *ideal* if there exists an ideal secret sharing scheme Σ realizing it.

An example of a perfect and ideal secret sharing scheme is the well-known Shamir's secret sharing scheme [112] for threshold access structures, i.e., access structures where any subset of size greater than k recovers the secret, while any subset of size less than k cannot.

Shamir's (k, n)-Threshold Secret Sharing Scheme

1. **Initialization.** The dealer chooses n distinct, non-zero elements of Z_p, $x_1, \ldots, x_n$, (where $p \geq n+1$). For $i = 1, \ldots, n$, the dealer assigns the value x_i to user i. The values x_i are public.
2. **Sharing.** Let $s \in Z_p$ be the secret the dealer wants to share. He secretly chooses (independently at random) $k-1$ elements of Z_p, say $a_1, \ldots, a_{k-1}$.
3. For $i = 1, \ldots, n,$, the dealer computes $y_i = a(x_i)$, where

$$a(x) = s + \sum_{k-1}^{j+1} a_j x^j \bmod p.$$

4. For $i = 1, \ldots, n$, the dealer gives the share y_i to participant i.
5. **Reconstruction.** The secret $s = a(0)$ can be reconstructed by any subset of k participants, say $\{1, \ldots, k\}$ for example, by computing for $j = 1, \ldots, k$, the coefficients

$$b_j = \prod_{1 \leq s \leq k, s \neq j} \frac{x_s}{x_s - x_j},$$

and, then, the $\sum_{j=1}^{k} b_j y_j$.

It is possible to show that, any subset of $k-1$ participants, by pooling together their own shares, gets *absolutely no* information on the secret s [112].

Constructions for secret sharing schemes for general access structures were first given in [74] and, subsequently, in many other papers (see [119] for references).

At this point, we can describe the so-called KIO construction, due to the use of **K**PS and **I**SSS to construct a **O**ne-time BES.

KIO Construction. Let $\mathcal{B} = \{B_1, \ldots, B_\beta\}$ be a family of subsets of $\mathcal{U}$, and let ω be an integer. For each $1 \leq j \leq \beta$, suppose a Fiat-Naor scheme $(\leq |B_j|, \leq \omega)$ is constructed with respect to user set B_j. The secret values associated with the j-th scheme will be denoted s_{jC}, where $C \subseteq B_j$ and $|C| \leq \omega$. The value s_{jC} is given to every user in $B_j \setminus C$. Moreover, suppose that $\Gamma \subseteq 2^{\mathcal{B}}$ and there exists a Γ-ISSS defined on $\mathcal{B}$ with values in $GF(q)$. Let $\mathcal{F} \subseteq 2^{\mathcal{U}}$, and suppose the following two properties are satisfied:

$$\{B_j : i \in B_j\} \in \Gamma \text{ for every } i \in \mathcal{U} \quad \text{and } \{B_j : |F \cap B_j| \geq \omega + 1\} \notin \Gamma \text{ for every } F \in \mathcal{F}.$$

Then, we can construct a $(\leq n, \mathcal{F})$-BES as follows: let $P \in \mathcal{U}$. The dealer can broadcast a message $m_P \in GF(q)$ to P using the following algorithm:

KIO Construction

1. For each $B_j \in \mathcal{B}$ the dealer computes a share $y_j \in GF(q)$ corresponding to the secret m_P.
2. For each $B_j \in \mathcal{B}$ the dealer computes the key k_j corresponding to the set $P \cap B_j$ in the Fiat-Naor scheme implemented on B_j:

$$k_j = \sum_{C \subseteq B_j : C \cap P = \emptyset, |C| \leq \omega} s_{jC}$$

3. For each $B_j \in \mathcal{B}$ the dealer computes $b_j = y_j + k_j$.
4. The broadcast is $b_P = (b_j : B_j \in \mathcal{B})$.

The basic idea of the KIO construction can be explained as follows: first, consider a user $i \in P$ and define $A_i = \{j : i \in B_j\}$. User i can compute k_j for every $j \in A_i$. Then, for each $j \in A_i$, i can compute $y_j = b_j - k_j$. Finally, since $A_i \in \Gamma$, i can compute the message m_P from the shares y_j where $j \in A_i$. On the other hand, let $F \in \mathcal{F}$ be such that $F \cap P = \emptyset$. Define

$$A_F = \{j : |F \cap B_j| \geq \omega + 1\}.$$

The coalition F can compute k_j, and hence y_j for every $j \in A_F$. However, they can obtain no information about the shares y_j, where $j \notin A_F$. Since $A_F \notin \Gamma$, F has no information about the value of m_P.

For other papers concerning with broadcast encryption the reader is referred to [18,27,58,22,28,30,65,76,81,85], to name a few.

4 Use of a Trusted Third Party

Another important approach to solve the key establishment problem requires an *on-line* Trusted Third Party, usually referred to as the *Key Distribution Center*. In this section we discuss the main advantages/disadvantages related to this approach, outlining the structures of some of the most common protocols.

4.1 Key Distribution Center

A common solution to the key establishment problem relies on the use of a *trusted party*, usually referred to as the *Key Distribution Center* (KDC, for short), responsible for the generation and the distribution of the keys to the users. In such a model, every user of the system is connected to the KDC by means of a private channel. When 2 or more users wish to privately communicate, one of them sends a key-request to the KDC. Then, the KDC generates at random a key κ and sends in a secure way κ to the users. Later on, the users can privately communicate by using κ.

This approach was initiated by Needhman and Schroeder [100]. The protocol they proposed can be described as follows: Let $\mathcal{T}$ denote the KDC. *Alice* and *Bob* have public identifiers, id_A and id_B, and share a secret key with $\mathcal{T}$, say k_{AT} and k_{BT}, respectively. Moreover, let r_A and r_B be random numbers.

Needham-Schroeder Protocol

1. *Alice* sends the message (id_A, id_B, r_A) to $\mathcal{T}$.
2. $\mathcal{T}$ sends to *Alice* the message $\mathcal{E}_{k_{AT}}(r_A, id_B, k, \mathcal{E}_{k_{BT}}(k, id_A))$.
3. *Alice* sends $\mathcal{E}_{k_{BT}}(k, id_A))$ to *Bob*.
4. *Bob* sends $\mathcal{E}_k(r_B))$ to *Alice*.
5. *Alice* sends $\mathcal{E}_k(r_B - 1))$ to *Bob*.

Let us briefly explain the steps of the protocol. *Alice* starts by sending her identifier id_A, Bob's identifier id_B and a random value r_A to $\mathcal{T}$. This message is basically as a key-request. $\mathcal{T}$ replies with an encrypted message for *Alice* of the session key k and of a sub message, encrypted for *Bob*, containing the same session key k. Then, *Alice* forwards to *Bob* the part of the message generated by $\mathcal{T}$ for him. The last two messages they exchange are used to confirm they have computed the same key.

However, as subsequently pointed out, the protocol presents some problems: in step 2 the part of the message for *Bob* is unnecessarily double encrypted. Moreover, since *Bob* has no way to check if the key k obtained in step 3 is fresh, if the session key k is compromised, anyone can re-send message in step 3 and can correctly compute the message in step 5.

On the Needham-Schroeder protocol were based many different protocols. Among them, the most famous is surely the so-called *Kerberos* System [101]. The system was conceived in 1989 at the MIT and supports both entity authentication and key establishment using symmetric encryption and a third party.

Kerberos System (simplified version)

1. *Alice* sends the message (id_A, id_B, r_A) to $\mathcal{T}$.
2. $\mathcal{T}$ sends to *Alice* the message $(\mathcal{E}_{k_{BT}}(k, id_A, L), \mathcal{E}_{k_{AT}}(k, r_A, L, id_B))$.
3. *Alice* sends $\mathcal{E}_{k_{BT}}(k, id_A, L), \mathcal{E}_k(id_A, t_A)$ to *Bob*.
4. *Bob* sends $\mathcal{E}_k(t_A)$ to *Alice*.

Notice that the structure is quite close to the structure of the Needham-Schroeder scheme. The main difference is the use of a *life-time* period L for the session key, and of a time stamp t_A of *Alice*'s clock. The value L enables to partially avoid the attack described for the Needham-Schroeder scheme. A full description of the Kerberos system can be found in [101], while for other on-line KDC-based schemes the reader is referred to [94] and to the references therein quoted. We just wanted to point out this approach by sketching two of these schemes, without going into details that are however of great importance in actual implementations.

Most of the protocols which use a KDC are "proved" to be secure by means of empiric arguments: the protocol are strong enough to deal with well-known attack strategies. Bellare and Rogaway [9] formally studied the KDC-based approach to the key establishment problem. In their paper [9], they proposed a formal three-party model, and described protocols with security proofs into the so called *random oracle model* [11].

Advantages of Session Keys. The use of a $\mathcal{TA}$ to solve the key establishment problem is particularly suitable due to the possibility of using session keys. A session key is a short-term key, usable for a restricted period of time, after which it is destroyed. Many reasons motivate session keys. Basically:

- *Ciphertext attacks.* If the key is used in a symmetric cryptosystem, the amount of ciphertext an adversary can use in order to break the scheme is limited.
- *Breaks in.* If the key is compromised, only data protected during the previous period are potentially exposed.
- *Memory Storage.* To reduce the number of secret keys that users have to store: session keys can generated when needed.

Notice that the use of a KDC is a suitable solution to key establishment, since, apart from the "pure distribution" of keys to users, several related key-management aspects (i.e., life time, authentication of the communicating entities, usage restrictions of a key and so on) can be easily solved with this third party. However, as we point out in the next subsection, the use of a KDC could cause some problems.

4.2 Distribution of a KDC

Our attention in this subsection focuses on a model which remedies some potential weaknesses introduced by using a *single KDC*. Indeed, the main drawback of a single KDC is that it works *on-line* and it must be *trusted*. Potentially, it could

eavesdrop all the communications. Moreover, the center can be a "bottleneck" for the performances of the network and, if it crashes, secure communication cannot be supported anymore. Last but not least, even if the KDC is honest and everything works fine, the KDC still represents an attractive target to the adversary. Indeed, the overall system security is lost if the KDC is compromised.

A frequently used solution to the availability problem lies in the *replication* of the KDC in various points of the network. This strategy reduces the communication delay which produces a single center but decreases the security of the overall system, since there are different physical locations which stores users' private keys that can be broken into. An adversary, which succeeds in controlling the center, can understand all the communications. A common solution for this problem consists in *partitioning* the network in various domains with dedicated KDCs, responsible of the key management only of a *fixed local area*. In a partitioned network, an adversary which controls the KDC of a domain has only power on a delimited part of the network.

However, partitioning of the network and replication of the KDC are partial and expensive solutions. The partition of a network implies an heavy communication overhead for inter-domain KDCs coordination in presence of key requests of groups of users which belong to different domains; while, replication of centers decreases security and introduces problems of consistence and synchronization between the servers during the update processes. As has been pointed out in [97], in a multi-cast communication environment with support for virtual meetings involving thousands of clients, and data streams transmission to a large group of recipients, the availability and security issues of a centralized environment become even more relevant and difficult to solve than with unicast communication.

A robust and efficient solution to the above issue could be a new approach to key distribution, introduced in [97]. A Distributed Key Distribution Center (DKDC, for short) is a set of n servers of a network that *jointly realize* the function of a Key Distribution Center. A user, who needs to communicate with a group of users, sends a key-request to a subset of his own choosing of the n servers, and the contacted servers answer with some information enabling the user to compute the common key. In such a model, a single server by itself does not know the secret keys, since they are *shared* among the n servers. Moreover, if some server crashes, secure communication can still be supported by the other servers and, since each user can contact a different subset of servers, the slowdown factor for the performances of the applications introduced by a single KDC can be improved.

The model we consider in this case is the following: Let $\mathcal{U} = \{U_1, \ldots, U_m\}$ be a set of m users, and let $S_1, \ldots, S_n$ be a set n servers of the network. Each user has *private connections* with all the servers. A scheme to set up a DKDC is divided in three phases: An *initialization phase*, which involves only the servers and requires (temporary) *private channels*; a *key request phase*, in which users ask for keys to servers; and a *key computation phase*, in which users retrieve keys from the messages received from the servers contacted during the key request phase. More precisely, the property that must hold are:

Properties of a DKDC

- When the initialization phase correctly terminates, each server S_i has to be able to compute some private information, denoted by a_i, enabling him to answer the key-request messages.
- Each user in a group $C_h \subseteq \mathcal{U}$ must be able to uniquely compute the group key, after interacting with at least k servers of his choice.
- A group key must be secure against attacks performed by coalitions of servers, coalitions of users, and hybrid coalitions (servers and users).

A construction for a DKDC, based on a family of ℓ-wise independent functions, has been proposed in [97]. A function is ℓ-wise independent if the knowledge of the value of the function in $\ell - 1$ different points of the domain does not convey any information on the value of the function in another point.

The scheme proposed in [97] enables ℓ groups of users, referred to as *conferences* in a set $\mathcal{C}$, *not known* a priori, to securely compute a common key. The family of ℓ-wise independent functions chosen in [97] to construct the scheme is the family of all bivariate polynomials $P(x, y)$ over a given finite field Z_q, in which the degree of x is $k - 1$ and the degree of y is $\ell - 1$. The protocol can be described as follows: Let k, n be two integers such that $k \leq n$, and let G be a coalition of users that could try to compute keys for conferences in which they do not belong to. Moreover, let $\ell = \max_{G \subseteq \mathcal{U}} \ell_G$ be the maximum number of conference keys that a coalition G of users can compute, and assume that the initialization phase is performed by the first k servers of the system. The full protocol can be described as follows:

Initialization Phase

- Each of the servers $S_1, \ldots, S_k$, performing the initialization phase, constructs a random bivariate polynomial $P^i(x, y)$ of degree $k - 1$ in x, and $\ell - 1$ in y by choosing $k \cdot \ell$ random elements in Z_q.
- Then, for $i = 1, \ldots, k$, server S_i evaluates the polynomial $P^i(x, y)$ in the identity j of S_j, and sends $Q_j^i(y) = P^i(j, y)$ to the server S_j, for $j = 1, \ldots, n$.
- For $j = 1, \ldots, n$, each server S_j computes his private information a_j, adding the k polynomials of degree $\ell - 1$, obtained from the k servers performing the initialization phase. More precisely,

$$a_j = Q_j(y) = \sum_{i=1}^{k} Q_j^i(y).$$

A user who needs a conference key, sends a key request to the servers as follows

KEY REQUEST PHASE

- A user $U \in C_h$, who wants to compute the key κ_h, sends to at least k servers, say $S_{i_1}, \ldots, S_{i_k}$, a request (U, h).
- Each server S_{i_j}, invoked by U, checks that the user belongs to C_h, and sends to U the value $Q_{i_j}(h)$, i.e., the value of the polynomial $Q_{i_j}(y)$ evaluated in $y = h$.

Finally, using the k values received from the servers $S_{i_1}, \ldots, S_{i_k}$, and applying the Lagrange formula for polynomial interpolation, each user $U \in C_h$ recovers the secret key $P(0, h) = \sum_{i=1}^{k} P^i(0, h)$. More precisely,

KEY COMPUTATION PHASE

- U computes, for $j = 1, \ldots, k$, the coefficients

$$b_j = \prod_{1 \le s \le k, s \ne j} \frac{i_s}{i_s - i_j}.$$

Then, he recovers $P(0, h)$ computing the $\sum_{j=1}^{k} b_j y_{i_j}$ where, for $j = 1, \ldots, k$, $y_{i_j} = Q_{i_j}(h)$, the value received from the server S_{i_j}.

The security of the above scheme is unconditional. However, in [97] some computationally secure constructions were given as well. Actually, the problem studied in [97] was a more general problem: how to securely distribute the computation of a pseudorandom function. A scheme for DKDC was considered as an applicative scenario for the distributed computation of a pseudorandom function.

Maurer, in his survey on future prospectives for Cryptography [92], has pointed out that two important directions for the research during the next years could be the weakening of the assumptions on which cryptographic protocols are built on, and the *distribution of trustiness*. Key Establishment is an important theoretical and practical problem, and distributed solutions seem to be suitable in many settings. This is the reason[3] for which we have included a paragraph to talk about the distribution of a KDC [97].

5 Multicast Schemes

Multicast communication schemes enable delivering data to multiple recipients. The motivation for such communication scheme lies in its efficiency: users of the same group get the same message simultaneously, with a consequent reduction of both sender and network resources. A wide range of applications benefit from

[3] Well, a less impartial reason is that we like this problem, and we have even studied some extensions [24,25,49] of the model given in [97].

multicast communication. However, several issues must be solved when designing a secure multicast scheme. The reader is referred to [43] for a clear and detailed overview.

Among them, one of the most challenging problem is the so called *access control*: only legitimate members of a multicast group must have access to the multicast group communication. The standard technique that is used to guarantee such requirement is to maintain a *common key* that is known to all the multicast group members, but is *unknown* to non-member.

In this setting, hence, the key establishment problem is how to maintain the invariant that all the group members, and only them, have access to a group key in a group with *dynamic membership*. Indeed, from time to time, users can be added to and removed from the group. This is the main difference between this setting and the previous ones, where groups are static (i.e., broadcast schemes).

The scenario we consider can be formalized as follows: Let $\mathcal{U}$ be the universe of all possible users, and let GC denote the group controller, responsible for the key-management problem. Let $M = \{u_1, \ldots, u_n\} \subseteq \mathcal{U}$ be the *multicast group*. We assume that $GC \notin M$. A session key k_s is shared initially by M and the GC. Moreover, other information and key material can be known by the users in M and the GC. The group M can change by means of two operations: *Join* and *Remove*. More precisely, let $U \subseteq M$. We have:

- *Remove(U)*. The new group is $M \setminus U$.
- *Join(U)*. The new group is $M \cup U$.

A *multicast re-keying protocol* specifies an algorithm by means of which the GC may update the session key k_s, and possible other information and key material held by the parties, after each *Join* and *Remove* operation.

The efficiency of such schemes is measured by means of:

- *Communication Complexity*. This parameter is the most important one, since reducing communication and network resources is the main motivation for multicast communication.
- *Group Controller Storage*. Amount of memory needed to manage the key-establishment issue.
- *User Storage*. Amount of memory the user needs to update the session keys for the multicast group.

To give an idea to the reader, we describe two multicast re-keying protocols: A basic scheme with minimal storage requirement, but inefficient from the communication complexity point of view, and a tree-based scheme, which improves the communication complexity paying something in terms of memory storage. The first one can be described as follows:

Storage Efficient Multicast Scheme

- Each user u holds the session key k_s, and a unique symmetric key k_u, shared with the GC. These keys are generated by GC in set up phase: for each user u, $k_u = f_r(u)$, where f is a pseudo-random function and r is a secret seed stored by GC.
- When a group of users U is removed from the group, GC chooses a new session key $k_s^{'}$, and sends it to the user u, by broadcasting the ciphers $E_{k_u}(k_s^{'})$ for all $u \in M \setminus U$.
- When a group of users U joins the group, GC generates a new session key $k_s^{'}$, and sends it to the new users, by broadcasting the ciphers $E_{k_u}(k_s^{'})$ for all $u \in U$, and to the old ones by broadcasting the cipher $E_{k_s}(k_s^{'})$.

The second scheme is based on a tree data structure. It enables a more efficient implementation of the update after a remove operation, and can be described as follows (we consider only the remove operation):

Tree-Based Multicast Scheme

- Let $n = 2^r$ (power of 2) be the number of users. The Group Controller GC sets up a binary tree of height $\log n$. Users are associated to the leaves. Then GC associates a key k_v to every node of the tree, and sends to each user through a secure channel the keys associated to the nodes along the path connecting the user to the root. The key associated to the root is the session key.
- When a user u must be removed from the group, GC performs the following operations: for each node v along the path from u to the root, a new key $k_v^{'}$ is generated. Then, these new keys are encrypted and broadcasted to the users. More precisely, denoting by $p(u)$ the parent of u and by $s(u)$ the sibling, $k_{p(u)}^{'}$ is encrypted with $k_{s(u)}$. The process is iterated until the root is reached.

The above scheme, described in [128], was subsequently improved by using a pseudo-random generator in [43], and further optimized, in order to improve the tradeoff between Center Memory Storage and Communication Complexity in [44]. In the latter paper lower bounds on the resources required by multicast schemes are given as well. Later on, in [104] it was shown that the trade-off constructions given in [44] are optimal.

6 Tracing Schemes

Digital valuable content can be distributed to a large set of parties by means of several media: cable or satellite networks, CD-ROM and DVD devices and more. If the content must be available only to authorized parties, namely the

ones that pay to get access, then it can be distributed in encrypted form, and the authorized users can receive decryption keys. The pay-per-view or certain subscription television broadcast transmissions are remarkable examples of such kind of content delivery systems.

However, the content is protected from forbidden users as long as they do not get decryption keys and, unfortunately, several reasons can drive authorized users, called *traitors*, to disclose/communicate their keys to other users, in order to enable them to access the data. In the pay-per-view scenario, for example, the decoder used to decrypt the transmission is a box storing some keys that are used, at the beginning of each transmission, to decrypt preliminary messages, sent by the broadcaster, enabling the reconstruction of the session key with which the subsequent content, say a movie, will be encrypted.

Several traitors can try to set up a new decoder by using subsets of their own key-sets. The new set does not belong to any user at all. Hence, in some way, nobody is guilty. Such phenomenon is called *Piracy*. Of course, if it is possible to *prove* that the decoder could be set up only because at least one of the users released some of his decryption keys, piracy can be prevented: if the risk to be accused is high, traitors can be discouraged.

Clearly, a possible solution is to encrypt the data separately under different keys, one for each user. This means that the total length of the ciphertext is at least n times the length of the cleartext, where n is the number of authorized parties. Such overhead is impossible in any broadcast environment.

In the recent years, researchers have concentrated their efforts on the design of systems preventing traitors from distributing the keys that enable the decryption of the encrypted content. The reader is referred to [48], which is the journal version of [47], where the concept of *tracing traitors* was introduced, and of [98], where some more efficient construction were given, for a complete introduction. This subsection is manly based on the treatment therein provided.

We would like to point out that the problem is related to the key establishment problem: as we show, several solutions are based on a smart distribution/allocation of decryption keys among the decoders, enabling to identify at least one traitors, once a pirate decoder is built by several traitors and captured.

The model we consider is the following: We have a data supplier $\mathcal{D}$ and a large set of recipients. The data supplier generates a meta-key which contains a base set A of random keys, and assigns subsets of these keys to users, m keys per user. These m keys form the user personal key. Different personal keys may have a nonempty intersection. We denote the personal key for user u by $P(u)$, which is a subset of the base set A.

A message in a traitor tracing scheme is a pair *(enabling block, cipher block)*. The cipher block is the symmetric encryption of the actual data, under some secret key s. The enabling block allows authorized users to obtain s. Basically, the enabling block consists of encrypted values under some or all of the keys of the base set A. Every authorized user is able to compute s by decrypting the values for which he has keys and then computing the actual key from these values.

The goal of the system designer is to assign keys to the users such that when a pirate decoder is captured it should be possible to detect at least one traitor, subject to the condition that the number of traitor is at most k. Such schemes are said to be *k-resilient.*

To exemplify the above concepts and to give to the reader an idea of what is going on, we describe two schemes. The first one, is very simple and is 1-resilient. It works as follows:

1-*resilient Traitor Tracing Scheme*

- The data supplier $\mathcal{D}$ generates $r = 2\log n$ keys

$$\{a_1^0, a_1^1, a_2^0, a_2^1, \ldots, a_{\log n}^0, a_{\log n}^1\}.$$

- Each user has a $\log n$ bits identity, and the personal key $P(i)$ for user i is the set of $m = \log n$ keys

$$\{a_1^{b_1}, a_2^{b_2}, \ldots, a_{\log n}^{b_{\log n}}\},$$

  where b_j is j-th bit in i's identity.
- Let s be the key used to encrypt the cipher block. The data supplier splits s into $\log n$ secrets $s_1, \ldots, s_{\log n}$, i.e., s is given by the XOR of the s_i, and encrypts every s_i with both a_i^0 and a_i^1. Both encryptions are added to the enabling block.

Notice that every user can decrypt the s_i and compute s. Different users have at least one row where they differ in the selected keys. Since any pirate decoder must contain at least a key for every $i = 1, \ldots, \log n$, and we assume that at most one traitor is allowed, then the pirate decoder must store exactly the keys of the traitor, which uniquely identify himself.

An efficient scheme and with higher resilience can be constructed by using a set of ℓ (unkeyed) hash functions.

k-resilient Traitor Tracing Scheme

- Let $\{h_1, \ldots, h_\ell\}$ be a set of hash functions chosen at random. Each function h_i maps $\{1, \ldots, n\}$ to $\{1, \ldots, 2k^2\}$. The data supplier $\mathcal{D}$ generates a matrix of $\ell \times 2k^2$ random keys, where each row is given by

$$A_i = \{a_{i,1}, a_{i,2}, \ldots, a_{i,2k^2}\}.$$

- Each user u receives a personal key

$$P(u) = \{a_{1,h_1(u)}, a_{2,h_2(u)}, \ldots, a_{\ell,h_\ell(u)}\}.$$

- Let s be the key used to encrypt the cipher block. The data supplier splits s into $2k^2$ secrets $s_1, \ldots, s_{2k^2}$, i.e., s is given by the XOR of the s_i, and encrypts every s_i with all the keys of row A_i. These encryptions are added to the enabling block.

Again, every authorized user recovers the secret key s. The tracing property can be obtaining by an appropriate choice of the set of hash functions. In such a case, if a pirate decoder is captured, the tracing algorithm simply identifies the highest number of keys that belong to a certain user. With high probability this user is one of the traitors. On the other hand, the probability that an innocent is accused is very small. Details can be found in [48].

Since the first paper on tracing traitors [47], many results have been achieved in this field, that has received attention from a large number of researchers. Some references about tracing (and multicast) schemes for the interested reader, just to name a few, are [17,32,34,48,57,59,103,65,98,99,96,71,72,73,78,108,109,116], [123,124].

7 Quantum Key Distribution

To close our quick overview about key distribution schemes, we would like to spend some words on quantum cryptography and, more precisely, on quantum key distribution. The reader is strongly encouraged to read the survey article by Gottesman and Lo [68] for a concise, simple and interesting introduction to the subject and its possible future prospectives.

During the last century, scientists have shown that classical physics is a powerful theory to describe the macroscopic world but almost useless for the microscopic one: here, the *determinism* of classical physics does not work in order to describe the intrinsically *random* behaviors of the particles. Moreover, in the microscopic world, Heisenberg's uncertainty principle, imposes a fundamental limitation to "the accuracy" of the measurements that can be done.

Quantum Information Processing is a new emerging research field in which people are studying the possibility of using quantum systems and quantum laws in information processing. Many efforts have been done in the recent years, and several difficult problems in the classical information processing scenario have been shown to be easily solvable in the quantum setting: if a quantum computer can be built, many public key cryptography schemes, for example, would be completely useless [113].

In Cryptography, apart from the destructive aspects related to possible applications of quantum algorithms and systems, some positive results have been achieved as well. One of the most remarkable is a method enabling two parties, which share a *quantum channel* and a public *classic channel*, to establish a common secret key for subsequent cryptographic uses. Bennett and Brassard proposed the first scheme in 1984 [16]. Nowadays, several groups have implemented and experimented quantum key distribution schemes, and some companies have even started their own businesses on these products (e.g., [68]).

Staying far from a precise and in-depth presentation, in the following we would like just to sketch how quantum key distribution works.

The key *Alice* wishes "to send" to *Bob* is a sequence of bits. The value of each bit is encoded on the properties of a photon, its polarization for example. The polarization is the oscillation direction of its electric field. Four possible polarizations are considered to represent the bits: vertical, horizontal, or diagonal.

Graphically, these polarizations can be represented by the symbols $\leftrightarrow, \updownarrow, \nearrow, \nwarrow$. *Alice* and *Bob* agree that $\leftrightarrow$ and $\nearrow$ represent 0, while $\updownarrow$ and $\nwarrow$ represent 1. A *filter* can be used to distinguish between horizontal $\leftrightarrow$ and vertical $\updownarrow$ photons; another one, between diagonal $\nearrow$ and $\nwarrow$ photons. Hence, each filter enables reading a photon which can encode zero or one.

The main property on which quantum key distribution is based on is that: *When a photon passes through the correct filter, its polarization does not change; while, if it passes through the wrong one, its polarization is modified randomly.*

For example, if a vertical or horizontal photon passes through the filter to distinguish between vertical and horizontal photons, its polarization does not change. Vice versa, if it passes through the filter to distinguish between diagonal photons, it randomly changes its polarization.

Basically the scheme works as follows: *Alice*, for each bit of the key, chooses a photon with one of the two possible polarizations to represent that bit and sends it, through the quantum channel to *Bob*. At each transmission, *Bob* chooses uniformly at random a filter to read horizontal and vertical photons or diagonal photons. At the end, he tells *Alice* his choices and *Alice* confirms the right ones. The bits read correctly by *Bob* form the basis for the common secret key. Indeed, in order to extract a common secret key from the sequence of bits, they have to check the absence of transmission errors and of *Eve*'s eavesdropping.

Roughly speaking, the security of the scheme is guaranteed since, if *Eve* tries to read the photons transmitted by *Alice* along the quantum channel, then on average half of the times she changes their polarizations! In this case, at the end of the quantum key distribution protocol, *Alice* and *Bob* can recognize her presence. In other words, we can even say that the laws of nature guarantee that an eavesdropper will either reveal itself with near certainty or gain no information about the key. The probability that an eavesdropper is not detected and nevertheless gains a substantial amount of information can be made as small as desired[4].

More precisely, but without going into the details, the protocol can be described as follows (see next page).

Notice that, even if *Eve* eavesdrops the communication that takes place over the public channel in step 3, she cannot figure out any information about the bits read by *Bob*, since each filter enables to read a photon which can encode zero or one.

At a first look, the scheme can be considered as a key transport scheme, since *Alice* chooses the initial sequence of bits: but, actually, the final key is the results of the random choices of *Bob* as well. Hence, if *Alice* chooses the string uniformly at random, even if the final key is a subset of the initial string, it is a random string generated by the random choices of both users. Hence, it can be better considered as a key agreement scheme.

[4] Notice that, even if intuitively simple, the formal proof of security of a quantum key distribution scheme is a very difficult task, due to the variety of quantum tricks that *Eve* can apply and that must be taken into account.

Quantum Key Distribution Scheme

1. For each key bit, *Alice* sends a photon, whose polarization is randomly selected. She records these polarizations/orientations.
2. For each incoming photon, *Bob* chooses randomly one of the two filters. He writes down his choice as well as the value he records.
3. After all photons have been transmitted, *Bob* reveals, over a conventional and unsecure channel - the phone line for example - to *Alice* the sequence of filters he used.
4. *Alice* tells *Bob* in which cases he chose the correct filter.
5. *Alice* and *Bob* now know in which cases their bits should be identical (when *Bob* used the correct filter). A subset of these bits will form the final key.
6. Finally, *Alice* and *Bob* check the common sequence of bits they hold. In this step error correcting codes are used and some bits are discarded. The remaining ones constitute the common secret key.

A drawback of the above scheme is that it assumes that, before running the protocol, *Alice* and *Bob authenticate each other* in some way (i.e., using some common information or some short shared key). The authentication is necessary to avoid an impersonation attack, where *Eve* pretends to be for example *Bob*. Hence, it cannot be used by two users that have never meet before.

A solution that can be used to solve the authentication problem is the introduction of a Quantum Cryptographic Center, universally known and trusted, that verifies the identity of both users.

Most experiments carried out up to now use optical fibers to implement the quantum channel, shared between *Alice* and *Bob*, to transmit the photons. Currently, distances up to 70 kilometers have been achieved at many places, for example, at Los Alamos (USA), at BT Labs (UK), at the University of Geneva (CH), and at the University of Vienna. However, experiments have even been conducted in Los Alamos in order to send the photons through the air. In this case, the ultimate goal is secure ground-to-satellite communication.

Finally, quantum key distribution is feasible with current technology, though at still rather low data rates (a few hundred bits per second).

8 Conclusions

Key Establishment is a vast topic. Perhaps, the uncovered aspects are more than the ones we have briefly mentioned in this paper. We have outlined some settings and protocols that seem to us to be representative of both problems and possible solutions. However, our aim was just to give a gentle introduction to the subject, mainly for students who approach the Key Establishment problem for the first time. Among important approaches that are totally missing from this version of the paper, the unconditionally secure key agreement technique by public discussion [90], surely would have deserved a whole section. We refer the reader to [90] for details and to [91] for papers on this approach and related techniques (e.g.,

privacy amplification). Even several variations of protocols for key distribution for dynamic groups, close in spirit to multicast schemes, supporting centralized and decentralized group control, should have been mentioned (e.g., [3,99]). The Key Escrow issue [62] and its practical/political implications should have been described, too (e.g., see [63] and the references therein quoted). As well as it would have been interesting to give a look at the world of the standards (e.g., [61]). For all these aspects we refer the reader to the proceedings of the major conferences in Cryptography (Crypto, Eurocrypt, and Asiacrypt) and to the journals involved in Cryptography and Theoretical Computer Science. Another good source of references, with notes about the history of the schemes, credits to the authors, and attributions of the results, can be found in the paragraphs at the end of Chapters 12 and 13 of [94]. What can we say more? If the reader has found the topic fascinating, and his curiosity is driving him to look for further papers, we have reached the goal for which we have been writing these pages: A 'quick introduction' is not needed anymore!

Acknowledgment

We would like to thank Doug Stinson for hints and suggestions, and Christof Zalka for references and comments on quantum key distribution.

References

1. M. Abdalla and M. Bellare *Increasing the Lifetime of a Key: A Comparative Analysis of the Security of Re-keying Techniques*, Advances in Cryptology - Asiacrypt '01, Lecture Notes in Computer Science, vol. 1976, pp. 546-559, 2001.
2. G. Agnew, R. Mullin, and S. Vanstone, *An Interactive Data Exchange Protocol Based on Discrete Exponentiation*, Advances in Cryptology - Eurocrypt '88, Lecture Notes in Computer Science, vol. 330, pp. 159-166, 1988.
3. J. Anzai, N. Matsuzaki, and T. Matsumoto, *A Quick Group Key Distribution Scheme with Entity Revocation*, Advances in Cryptology - Asiacrypt '99, Lecture Notes in Computer Science, Vol. 1716, pp. 333-347.
4. S. Bakhtiari, R. Safavi-Naini, and Josef Pieprzyk, *On password-based authenticated key exchange using collisionful hash functions*, Advances in Cryptology – Australasian Conference on Information Security and Privacy (ACISP '96), Lecture Notes in Computer Science, vol. 1172, pp. 298–309, 1996.
5. G. R. Blakley, *Safeguarding Cryptographic keys*, AFIPS Conference Proceedings, vol. 48, pp. 313-317, 1979.
6. A. Beimel and B. Chor, *Interaction in Key Distribution Schemes*, Advances in Cryptology - Crypto '93, Lecture Notes in Computer Science, vol. 773, pp. 444–455, 1994.
7. A. Beimel and B. Chor, *Communication in Key Distribution Schemes*. IEEE Transactions on Information Theory, N. **42**, 19–28, 1996.
8. M. Bellare, R. Canetti, and H. Krawczyk, *A modular approach to the design and analysis of authentication and key exchange protocols*, Proceedings of the 30th Annual Symposium on the Theory of Computing, ACM, pp. 419–428, 1998.

9. M. Bellare and P. Rogaway, *Provably Secure Session Key Distribution: The Three Party Case*, Proceedings of the 27th Annual Symposium on the Theory of Computing, ACM, pp. 57–66, 1995.
10. M. Bellare and P. Rogaway, *Entity Authentication and Key Distribution*, Advances in Cryptology - Crypto '93, Lecture Notes in Computer Science, Vol. 950, pp. 92-111, 1995.
11. M. Bellare and P. Rogaway, *Random Oracle are Practical: A Paradigm for Designing Efficient Protocols*, Proceedings of the 1*st* ACM Conference on Computer and Security, ACM Press, pp. 66-73, 1993.
12. M. Bellare, L. Cowen, and S. Goldwasser, *On the Structure of Secret Key Exchange Protocols*, Advances in Cryptology - Crypto '89, Lecture Notes in Computer Science, Vol. 435, pp. 604-605, 1989.
13. M. Bellare, D. Pointcheval and P. Rogaway, *Authenticated Key Exchange Secure Against Dictionary Attacks*, Advances in Cryptology - Eurocrypt 2000, Lecture Notes in Computer Science, Vol. 1807, pp. 139-155, 2000.
14. M. J. Beller and Y. Yacobi, *Minimal Asymmetric Authentication and Key Agreement Schemes*, unpublished manuscript, 1994.
15. M. J. Beller and Y. Yacobi, *Fully-Fledged Two-way Public Key Authentication and Key Agreement for Low-Cost Terminals*, Electronics Letters, Vol. 29, pp. 999-1001, 1993.
16. C. H. Bennett and G. Brassard, *Quantum Cryptography: Public Key Distribution and Coin Tossing*, Proceedings of IEEE International Conference on Computer Systems and Signal Processing, Bangalore India, pp. 175-179, 1984.
17. O. Berkman, M. Parnas, and J. Sgall, *Efficient Dynamic Traitor Tracing*, Proc. of the 11-th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), pp. 586–595, 2000.
18. S. Berkovits, *How to Broadcast a Secret*, Advances in Cryptology - Eurocrypt '91, Lecture Notes in Computer Science, vol. **547**, pp. 536–541, 1991.
19. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, *The Kryptoknight family of light-weight protocols for authentication and key distribution*, IEEE/ACM Transactions on Networking, vol. 3, no. 1, pp. 31-41, 1995.
20. Official site of Bletchley Park, *http://www.cranfield.ac.uk/ccc/bpark*
21. R. Blom, *An Optimal Class of Symmetric Key Generation Systems*, Advances in Cryptology - Eurocrypt '84, Lecture Notes in Computer Science, vol. 209, pp. 335–338, 1984.
22. C. Blundo and A. Cresti, *Space Requirements for Broadcast Encryption*, Advances in Cryptology - Eurocrypt '94, Lecture Notes in Computer Science, vol. 950, pp. 287–298, 1995.
23. C. Blundo, P. D'Arco, and A. Giorgiogaggia, *A τ-restricted Key Agreement Scheme*, The Computer Journal, Vol. 42, N.1, pp. 51–61, 1999.
24. C. Blundo, P. D'Arco and C. Padrò, *A Ramp Model for Distributed Key Distribution Schemes*, Discrete Applied Mathematics, to appear 2002.
25. C. Blundo, P. D'Arco, V. Daza and C. Padrò, *Bounds and Constructions for Unconditionally Secure Distributed Key Distribution Schemes for General Access Structures*, Proceedings of the Information Security Conference (ISC 2001), Lecture Notes in Computer Science, vol. 2200, pp. 1-17, 2001.
26. C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung, *Perfectly-Secure Key Distribution for Dynamic Conferences*, Information and Computation, vol. 146, no. 1, pp. 1–23, 1998.
27. C. Blundo, A. De Santis, and U. Vaccaro, *Randomness in Distribution Protocols*, Information and Computation, vol. 131, no. 2, pp. 111–139, 1996.

28. C. Blundo, L. A. Frota Mattos, and D. R. Stinson, *Tradeoffs Between Communication and Storage in Unconditionally Secure Schemes for Broadcast Encryption and Interactive Key Distribution*, Advances in Cryptology - Crypto '96, Lecture Notes in Computer Science, vol. 1109, pp. 387–400, 1996.
29. C. Blundo, Luiz A. Frota Mattos, and D. R. Stinson, *Generalized Beimel-Chor Schemes for Broadcast Encryption and Interactive Key Distribution*, Theoretical Computer Science, vol. 200, pp. 313–334, 1998.
30. C. Blundo, L. A. Frota Mattos, and D. R. Stinson, *Multiple Key Distribution Maintaining User Anonymity via Broadcast Channels*, Journal of Computer Security, N. 3, pp. 309–323, 1994/95.
31. V. Boyko, P. MacKenzie, and S. Patel *Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman*, Advances in Cryptology - Eurocrypt '00, Lecture Notes in Computer Science, vol. 1807, pp. 156–171, 2000.
32. D. Boneh and M. Franklin, *An Efficient Public Key Traitor Scheme*, Advances in Cryptology - Crypto '99, Lecture Notes in Computer Science, vol. 1666, pp. 338–353, 1999.
33. D. Boneh and R. J. Lipton, *Algorithms for Black-Box Fields and their Application to Cryptography*, Advances in Cryptology - Crypto '96, Lecture Notes in Computer Science, Vol. 1109, pp. 283-297, 1996.
34. D. Boneh and J. Shaw, *Collusion-Secure Fingerprinting for Digital Data*, IEEE Transactions on Information Theory, Vol. 44, No. 5, pp. 1897-1905, 1998.
35. D. Boneh and R. Venkatesan, *Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes*, in Advances in Cryptology - Crypto '96, Lecture Notes in Computer Science, vol. 1109, pp. 114–128, 1996.
36. G. Brassard and L. Salvail, *Secret-Key Reconciliation by Public Discussion*, Advances in Cryptology - Eurocrypt '93, Lecture Notes in Computer Science, Vol. 765, pp. 410-423, 1993.
37. E. Bresson, O. Chevassut and D. Pointcheval, *The Group Diffie-Hellman Problems*, Proceedings of SAC '02, Lecture Notes in Computer Science, 2002.
38. E. Bresson, O. Chevassut and D. Pointcheval, *Group Diffie-Hellman Key Exchange Secure Against Dictionary Attacks*, Advances in Cryptology - Asiacrypt '02, Lecture Notes in Computer Science, 2002.
39. E. Bresson, O. Chevassut and D. Pointcheval, *Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions*, In Advances in Cryptology - Eurocrypt '02 Lecture Notes in Computer Science vol. 2332, pp. 321-336, 2002.
40. E. Bresson, O. Chevassut and D. Pointcheval, *Provably Authenticated Group Diffie-Hellman Key Exchange: The Dynamic Case*, In Advances in Cryptology - Asiacrypt '01 Lecture Notes in Computer Science vol. 2248, pp. 290-309, 2001.
41. J. Buchmann, S. Dullmann, and H. Williams, *On the Complexity and Efficiency of a new key Exchange System*, Advances in Cryptology - Eurocrypt '89, Lecture Notes in Computer Science, vol. 434, pp. 597–616, 1989.
42. M. Burmester, *On the Risk of Opening Distributed Keys*, Advances in Cryptology - Crypto '94, Lecture Notes in Computer Science, Vol. 839, pp. 308–317, 1994.
43. R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, *Issue in Multicast Security: A Taxonomy and Efficient Constructions*, Infocom '99, pp. 708–716, 1999.
44. R. Canetti, T. Malkin, and K. Nissim, *Efficient Communication-Storage Tradeoffs for Multicast Encryption*, Advances in Cryptology - Eurocrypt '99, Lecture Notes in Computer Science, vol. 1592, pp. 459–474, 1999.

45. R. Canetti and H. Krawczyk, *Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels*, Advances in Cryptology - Eurocrypt '01, Lecture Notes in Computer Science, vol. 2045, pp. 453–474, 2001.
46. R. Canetti and H. Krawczyk, *Universally Composable Notions of Key Exchange and Secure Channels* Advances in Cryptology - Eurocrypt '02, Lecture Notes in Computer Science, vol. 2332, pp. 337–351, 2002.
47. B. Chor, A. Fiat and M. Naor, *Traitor Tracing*, Advances in Cryptology - Crypto '94, Lecture Notes in Computer Science, vol. 839, pp. 257–270, 1994.
48. B. Chor, A. Fiat, M. Naor and B. Pinkas, *Traitor Tracing*, IEEE Transactions on Information Theory, vol. 46, No. 3, pp. 893–910, May 2000.
49. P. D'Arco and D. R. Stinson, *On Unconditionally Secure Robust Distributed Key Distribution Centers*, Asiacrypt '02, to appear, 2002.
50. G. Davida, Y. Desmedt, and R. Peralta, *A key Distribution System Based on Any One-Way Function*, Advances in Cryptology – Eurocrypt '89, Lecture Notes in Computer Science, vol. 434, pp. 75-80, 1989.
51. G. Davida, Y. Desmedt, and R. Peralta, *On the importance of memory resources in the security of key exchange protocols*, in Advances in Cryptology – Eurocrypt '90, Lecture Notes in Computer Science, vol. 473, pp. 11-15, 1990.
52. D. E. Denning and G. M. Sacco, *Timestamps in key distribution protocols*, Communications of the ACM, Vol. 24, n. 8, pp. 533-536, 1991.
53. Y. Desmedt and M. Burmester, *Towards practical proven secure authenticated key distribution*, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, USA, pp. 228-231, 1993.
54. W. Diffie and M. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, N. 22, pp. 644–654, 1976.
55. W. Diffie, P. C. Van Oorschot, and M. J. Wiener, *Authentication and Authenticated Key Exchanges*, Design, Codes, and Cryptography, vol. 2, pp. 107-125, 1992.
56. M. Dyer, T. Fenner, A. Frieze and A. Thomas, *Key Storage in Secure Networks*, Journal of Cryptology, N. 8, pp. 189–200, 1995.
57. C. Dwork, J. Lotspiech, and M. Naor, *Digital Signets: Self-Enforcing Protection of Digital Information*, Proceedings of the 28-th Symposium on the Theory of Computation, pp. 489–498, 1996.
58. A. Fiat and M. Naor, *Broadcast Encryption*, Advances in Cryptology - Crypto '93, Lecture Notes in Computer Science, vol. 773, pp. 480–491, 1994.
59. A. Fiat and T. Tessa, *Dynamic Traitor Tracing*, Journal of Cryptology, Vol. 14, pp. 211–223, 2001.
60. M. Fischer and R. N. Wright, *Multiparty Secret Key Exchange Using a Random Deal of Cards*, Advances in Cryptology - Crypto '91, Lecture Notes in Computer Science, Vol. 576, pp. 141-155, 1991.
61. Federal Information Processing Standards Publications (FIPS), *http://www.itl.nist.gov/fipspubs/index.htm*
62. FIPS PUB 185, *Escrowed Encryption Standard*, 1994.
63. Y. Frankel and M. Yung, *Escrow Encryption Systems Visited: Attacks, Analysis, and Designs*, Advances in Cryptology - Crypto '95, Lecture Notes in Computer Science, vol. 963, p. 222–235, 1995.
64. E. Gafni, J. Staddon, and Y. L. Yin, *Efficient Methods for Integrating Traceability and Broadcast Encryption*, Advances in Cryptology - Crypto '99, Lecture Notes in Computer Science, vol. 1666, p. 372–387, 1999.

65. J. Garay, J. Staddon, and A. Wool, *Long-Lived Broadcast Encryption*, Advances in Cryptology - Crypto 2000, Lecture Notes in Computer Science, vol. 1880, pp. 333–352, 2000.
66. M. Girault, *Self-Certifying Public Keys*, Advances in Cryptology - Eurocrypt '91, Lecture Notes in Computer Science, vol. 547, pp. 490-497, 1991.
67. L. Gong and D. L. Wheeler, *A Matrix Key Distribution Scheme*, Journal of Cryptology, vol. 2, pp. 51–59, 1990.
68. D. Gottesman and H-K. Lo, *From Quantum Cheating to Quantum Security*, Physics Today on-line, available at http://www.aip.org/pt/vol-53/iss-11/p22.html
69. C.G. Gunther, *An Identity-Based Key-Exchange Protocol*, Advances in Cryptology - Eurocrypt '89, Lecture Notes in Computer Science, Vol. 434, pp. 29-37, 1990.
70. J. Katz, R. Ostrovsky, and M. Yung, *Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords*, Advances in Cryptology - Eurocrypt '01, Lecture Notes in Computer Science, Vol. 2045, pp. 475–494, 2001.
71. A. Kiayias and M. Yung, *Traitor Tracing with Constant Transmission Rate*, Advances in Cryptology - Eurocrypt '02, Lecture Notes in Computer Science, vol. 2332, pp. 450-465, 2002.
72. A. Kiayias and M. Yung, *Self Protecting Pirates and Black-Box Traitor Tracing*, Advances in Cryptology - Crypto '01, Lecture Notes in Computer Science, vol.2139 , pp. 63-79, 2001.
73. R. Kumar, S. Rajagopalan, and A. Sahai *Coding Constructions for Blacklisting Problems without Computational Assumptions*, Lecture Notes in Computer Science, vol. 1666, pp. 609–623, 1999.
74. M. Ito, A. Saito, and T. Nishizeki, *Secret Sharing Schemes Realizing General Access Structures*, IEEE Global Telecommunications Conference, pp. 99–102, 1987.
75. ITU-T REC. X.509 (Revised), *The Directory - Authentication Framework*, International Telecommunication Union, Geneva, Switzerland, July 1995.
76. M. Just, E. Kranakis, D. Krizanc and P. Van Oorschot, *On Key Distribution via True Broadcasting*, Proceedings of the 2nd ACM Conference on Computer and Communications Security, pp. 81–88, 1994.
77. D. Kahn, *The Codebreakers*, Scribner, New York, 1996.
78. H. Kim, D. H. Lee, M. Yung, *Privacy against Piracy: Protecting Two-Level Revocable P-K Traitor Tracing*, Lecture Notes in Computer Science ACISP, vol. 2384, pp. 482-496, 2002.
79. K. Koyama and K. Ohta, *Identity-based conference key distribution systems*, Advances in Cryptology - Crypto '87, Lecture Notes in Computer Science, vol. 917, pp. 175–185, 1987.
80. A. G. Konheim, *Cryptography: A Primer*, John Wiley & Sons, New York, 1981.
81. V. Korjik, M. Ivkov, Y. Merinovitch, A. Barg and H. Van Tilborg, *A Broadcast Key Distribution Scheme Based on Block Designs*, Lecture Notes in Computer Science, vol. 1025, pp. 12–21, 1995.
82. K. Kurosawa, K. Okada, and K. Sakano, *Security of the Center in Key Distribution Schemes*, Advances in Cryptology - Asiacrypt '94, Lecture Notes in Computer Science, vol. 917, pp. 333-341, 1995.
83. T. Leighton and S. Micali, *Secret key Agreement without Public Key Cryptography*, Advances in Cryptology - Crypto '93, Lecture Notes in Computer Science, vol. 773, pp. 456–479, 1993.
84. J. H. Van Lint and R. M. Wilson. (1992) *A course in combinatorics.* Cambridge University Press.

85. M. Luby and J. Staddon, *Combinatorial Bounds for Broadcast Encryption*, Advances in Cryptology - Eurocrypt '98, Lecture Notes in Computer Science, vol. 1403, pp. 512–526, 1998.
86. P. MacKenzie, S. Patel, and R. Swaminathan, *Password-Authenticated Key Exchange Based on RSA*, Advances in Cryptology - Asiacrypt '01, Lecture Notes in Computer Science, vol. 1976, pp. 599–613, 2001.
87. T. Matsumoto, *Incidence Structure for Key Sharing*, Advances in Cryptology - Asiacrypt '94, Lecture Notes in Computer Science, vol. 917, pp. 342–353, 1995.
88. T. Matsumoto, and H. Imai, *On the Key Predistribution System: A Practical Solution to the Key Predistribution Problem*, Advances in Cryptology - Crypto '87, Lecture Notes in Computer Science, vol. 293, pp. 185–194, 1987.
89. T. Matsumoto, Y. Takashima, and H. Imai, *On Seeking Smart Public-Key Distribution Systems*, Transactions of the IECE (Japan), Vol. 69, pp. 99-106, 1986.
90. U. Maurer, *Secret Key Agreement by Public Discussion*, IEEE Transaction on Information Theory, vol. 39, pp. 733-742, 1993.
91. ETH Crypto Group (Zurich), *http://www.crypto.ethz.ch/research/*
92. U. Maurer, *Cryptography* $2000_{+/-}10$, Advances in Cryptology, Lecture Notes in Computer Science, vol. 200, pp. 63–85, 2000.
93. U. Maurer and S. Wolf, *On the Complexity of Breaking the Diffie-Hellman Protocol*, SIAM Journal on Computing, vol. 28, pp. 1689-1721, 1999.
94. A.J. Menezes, P.C. Oorschot, and S.A. Vanstone, **Handbook of Applied Cryptography**, CRC Press, 1996.
95. C. J. Mitchell and F. C. Piper, *Key Storage in Secure Networks*, Discrete Applied Mathematics, vol. 21, pp. 215–228, 1988.
96. D. Naor, M. Naor and J. Lotspiech *Revocation and Tracing Schemes for Stateless Receivers* Advances in Cryptology - Crypto '01, Lecture Notes in Computer Science, vol. 2139, pp. 41–62, 2001.
97. M. Naor, B. Pinkas, and O. Reingold. *Distributed Pseudo-random Functions and KDCs*, Advances in Cryptology - Eurocrypt'99, Lecture Notes in Computer Science, vol. 1592, pp. 327–346, 1999.
98. M. Naor and B. Pinkas, *Threshold Traitor Tracing*, Advances in Cryptology - Crypto '98, Lecture Notes in Computer Science, vol. 1462, pp. 502–517, 1998.
99. M. Naor and B. Pinkas, *Efficient Trace and Revoke Schemes*, Financial Cryptography 2000, Lecture Notes in Computer Science, vol. 1962, pp. 1–21, 2000.
100. R. M. Needham and M. D. Schroeder. *Using Encryption for Authentication in Large Networks of Computers*, Communications of ACM, vol. 21, pp. 993–999, 1978.
101. B. C. Neuman and T. Tso. *Kerberos: An Authentication Service for Computer Networks*, IEEE Transactions on Communications, vol. 32, pp. 33–38, 1994.
102. C. Park, K. Kurosawa, T. Okamoto, and S. Tsujii, *On Key Distribution and Authentication in Mobile Radio Networks*, Advances in Cryptology - Eurocrypt '93, Lecture Notes in Computer Science, vol. 765, pp. 461–470, 1993.
103. B. Pfitzmann, *Trials of Traced Traitors*, Information Hiding, Lecture Notes in Computer Science, vol. 1174, pp. 49-64, 1996.
104. R. Poovendran, J. S. Baras, *An Information Theoretic Analysis of Rooted-Tree Based Secure Multicast Key Distribution Schemes*, Advances in Cryptology, Crypto '99, vol. 1666, pp. 624-638, 1999.
105. K. A. S. Quinn, *Some Constructions for Key Distribution Patterns*, Designs, Codes and Cryptography, vol. 4, pp. 177–191, 1994.
106. R. Rivest, *Cryptography*, Chapter 13 in *Handbook of Theoretical Computer Science*, (J. van Leeuwen, ed.) MIT Press, 1990.

107. R. Rivest, A. Shamir, and L. Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of ACM, Vol. 21, pp. 120-126, 1978.
108. R. Safavi-Naini and H. Wang, *New Constructions for Multicast Re-Keying Schemes Using Perfect Hash Families*, 7th ACM Conference on Computer and Communication Security, ACM Press, pp. 228–234, 2000.
109. R. Safavi-Naini and Y. Wang, *Sequential Traitor Tracing*, Lecture Notes in Computer Science, vol. 1880, p. 316–332, 2000.
110. R. Scheidler, J. A. Buchmann, and H. C. Williams, *Implementation of a key exchange protocol using some real quadratic fields*, Advances in Cryptology – Eurocrypt '90, Lecture Notes Computer Science, vol. 473, pp. 98-109, 1990.
111. S. Sing, *The Code Book: The Evolution of Secrecy from Mary Queen of Scots to Quantum Cryptography*, 1999.
112. A. Shamir, *How to Share a Secret* Communications of ACM, vol. 22, n. 11, pp. 612–613, 1979.
113. P. W. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM Journal on Computing, n. 26, pp. 1484-1509, 1997.
114. V. Shoup, *Lower Bounds for Discrete Logarithms and Related Problems*, Advances in Cryptology - Eurocrypt '97, Lecture Notes in Computer Science, Vol. 1233, pp. 256-266, 1997.
115. V. Shoup and A. Rubin, *Session Key Distribution Using Smart Cards*, Advances in Cryptology - Eurocrypt '96, Lecture Notes in Computer Science, vol. 1070, pp. 321-332, 1996.
116. J. N. Staddon, D.R. Stinson and R. Wei, *Combinatorial properties of frameproof and traceability codes*, IEEE Transactions on Information Theory vol. 47, pp. 1042-1049, 2001.
117. M. Steiner, G. Tsudik and M. Waidner, *Diffie-Hellman Key Distribution Extended to Groups*, Proceedings of the 3-rd ACM Conference on Computer and Communications Security, pp. 31–37, 1996.
118. M. Steiner, G. Tsudik and M. Waidner, *Key Agreement in Dynamic Peer Groups*, IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 8, pp. 769-780, 2000.
119. D. R. Stinson, *An Explication of Secret Sharing Schemes*, Designs, Codes and Cryptography, Vol. 2, pp. 357-390.
120. D.R. Stinson, **Cryptography: Theory and Practise**, CRC Press, 1995 (2nd Edition, 2002).
121. D. R. Stinson, *On Some Methods for Unconditionally Secure Key Distribution and Broadcast Encryption*, Designs, Codes and Cryptography, vol. 12, pp. 215–243, 1997.
122. D. R. Stinson and T. van Trung, *Some New Results on Key Distribution Patterns and Broadcast Encryption*, Designs, Codes and Cryptography, vol. 15, pp. 261–279, 1998.
123. D. R. Stinson and R. Wei, *Key preassigned traceability schemes for broadcast encryption*, Proceedings of SAC'98, Lecture Notes in Computer Science, vol. 1556, pp. 144-156, 1999.
124. D. R. Stinson and R. Wei, *Combinatorial properties and constructions of traceability schemes and frameproof codes*, SIAM Journal on Discrete Mathematics, vol. 11, pp. 41–53, 1998.

125. P. Syverson and C. Meadows, *Formal Requirements for Key Distribution Protocols*, Advances in Cryptology - Eurocrypt '94, Lecture Notes in Computer Science, vol. 950, pp. 320-331, 1994.
126. W. Tzeng and Z. Tzeng, *Round-Efficient Conference Key Agreement Protocols with Provable Security*, Advances in Cryptology - Asiacrypt '01, Lecture Notes in Computer Science, vol. 1976, pp. 614-628, 2001.
127. P. C. van Oorschot and M. J. Wiener, *On the Diffie-Hellman Key Agreement with Short Exponents*, Advances in Cryptology - Eurocrypt '96, Lecture Notes in Computer Science, Vol. 1070, pp. 332-341, 1996.
128. D. M. Wallner, E. J. Harder, and R. C. Agee, *Key Management for Multicast: Issues and Architectures*, Internet Draft (draft-wallner-key-arch-01.txt), ftp://ftp.ieft.org/internet-drafts/draft-wallner-key-arch-01.txt.
129. D. S. Wong and A. H. Chan, *Efficient and Mutually Authenticated Key Exchange for Low Power Computing Devices*, Advances in Cryptology - Asiacrypt '01, Lecture Notes in Computer Science, vol. 2248, pp. 272–281, 2001.
130. Y. Yacobi, *A Key Distribution Paradox*, Advances in Cryptology - Crypto '90, Lecture Notes in Computer Science, vol. 537, pp. 268–273, 1990.
131. Y. Yacobi and Z. Shmuely, *On key Distribution Systems*, Advances in Cryptology - Crypto '89, Lecture Notes in Computer Science, vol. 435, pp. 344–355, 1989.
132. Y. Zheng, *How to Break and Repair Leighton and Micali's Key Agreement Protocol*, Advances in Cryptology - Eurocrypt '94, Lecture Notes in Computer Science, Vol. 950, pp. 92-111, 1994.

A Survey of Name-Passing Calculi and Crypto-Primitives*

Michele Bugliesi[1], Giuseppe Castagna[2], Silvia Crafa[1], Riccardo Focardi[1], and Vladimiro Sassone[3]

[1] Università "Ca' Foscari", Venezia
[2] Ecole Normale Superieure, Paris
[3] University of Sussex

Abstract. The paper surveys the literature on high-level name-passing process calculi, and their extensions with cryptographic primitives. The survey is by no means exhaustive, for essentially two reasons. First, in trying to provide a coherent presentation of different ideas and techniques, one inevitably ends up leaving out the approaches that do not fit the intended roadmap. Secondly, the literature on the subject has been growing at very high rate over the years. As a consequence, we decided to concentrate on few papers that introduce the main ideas, in the hope that discussing them in some detail will provide sufficient insight for further reading.

Outline of the Paper

We start in Section 1 with a brief review of a polyadic version of Milner's π-calculus. Then we outline the foundational work by Pierce and Sangiorgi on typing systems for the π-calculus. Section 3 covers the Join Calculus, and a discussion on its type systems. The remaining sections cover security specific extensions of name-passing calculi. In Section 4 we review an extension of the π-calculus with a new construct for group creation, and study the impact of the new primitive in enforcing secrecy. In Section 5 we discuss the *security π-calculus*, a typed version of the asynchronous π-calculus, which applies type based techniques provide security resource access control and information flow security guarantees. Section 6 gives a brief outline of a value passing extension of CCS, known as CryptoSPA, with cryptographic primitives. Finally, Section 7 covers the spi-calculus, and its typing system(s) for secrecy. Each section includes pointers to further important work in the literature relevant to each of the topics.

1 The Pi Calculus

The π-calculus is a way of describing and analyzing systems consisting of agents which interact among each other, and whose configuration is continually changing. The π-calculus emerged as the canonical model of concurrent computation, in much the same way as the λ-calculus has established itself as the canonical model of functional computation.

* Research supported by 'MyThS: Models and Types for Security in Mobile Distributed Systems', EU FET-GC IST-2001-32617.

R. Focardi and R. Gorrieri (Eds.): FOSAD 2001/2002, LNCS 2946, pp. 91–138, 2004.

The λ-calculus emphasizes the view of computation as the process of taking arguments and yielding results. In the λ-calculus everything is a function, and computation is, essentially, the result of function application. *Concurrent* computation cannot be forced into this functional metaphor of computation without severe distortions: if anything, functional computation is a special case of concurrent computation, and one should reasonably expect to find the functional model represented within a general enough model of concurrency.

In the π-calculus, every term denotes a process – a computational activity running in parallel with other processes and possibly containing several independent subprocesses. Computation arises as a result of process interaction, which in turns is based on communication on named channels. Naming is, in fact, the pervasive notion of the calculus, for various reasons. Naming presupposes *independence*: one naturally assumes that the *namer* and the *named* are independent (concurrent) entities. Further, using a name, or address, is a prerequisite to the act of *communicating*, and of locating and modifying data.

Based on these observations, the π-calculus seeks ways to treat data-access and communication as the same thing: in doing so, it presupposes that naming of *channels* is primitive, while naming of *agents* is not. As we shall see, departing from this view, and extending the concept of naming to agents and *locations* is what led to the development of models of mobility on top of the π-calculus. As of now, however, we start looking at the π-calculus in itself.

1.1 Syntax and Operational Semantics

There are in fact several versions of the π-calculus. Here, we will concentrate on a very basic one, although polyadic: the differences with other versions are mostly orthogonal to our concerns. The syntax is given in Table 1.

We assume an infinite set of *names* to be used for values and communication channels, and an infinite set of variables. We let $a, b - p, q$ range over names and $x - z$ range over variables. In addition, we often reserve u and v to denote names or variables indistinguishably, whenever the distinction between the two notions may safely be disregarded.

We use a number of notation conventions: $\tilde{x} : \tilde{T}$ stands for $x_1 : T_1, \ldots, x_k : T_k$, and we omit trailing dead processes, writing $\overline{u}\langle N \rangle$ for $\overline{u}\langle N \rangle.\mathbf{0}$ and $u(\tilde{x} : \tilde{T})$ for $u(\tilde{x} : \tilde{T}).\mathbf{0}$. The empty tuple plays the role of synchronization messages. The input prefix and the restriction operator are binders: the notations $fn(P)$ and $fv(P)$ indicate, respectively, the set of free names and free variables of the process P: these notions are defined as usual. We assume identity for α-convertible terms throughout, and we often omit type annotations on the two binders whenever irrelevant to the context in question.

The syntactic form $\mathbf{0}$ denotes the inert process, which does nothing. $u(x : T)P$ is a process that waits to read a value on the channel u: having received a value, say M, it behaves as P with every free occurrence of x substituted by M. Dually, $\overline{u}\langle M \rangle.P$ is a process that sends a value M on channel u and then behaves as P. The syntax suggests that output, as input, is *synchronous*, hence blocking: before continuing as P

Table 1 Pi calculus (typed) syntax

Expressions M,N	$::=$	bv	basic value
	$\mid$	$a,\ldots,p$	name
	$\mid$	$x,\ldots,z$	variable
	$\mid$	$(M_1,\ldots,M_k)$	tuple, $k \geqslant 0$
Processes P,Q,R	$::=$	$\mathbf{0}$	stop
	$\mid$	$\overline{u}\langle N\rangle.P$	output
	$\mid$	$u(\tilde{x}:\tilde{T}).P$	input
	$\mid$	$(\nu\, a:T)P$	restriction
	$\mid$	$P \mid P$	composition
	$\mid$	$!P$	replication

the process the output $\overline{u}\langle M\rangle$ must be consumed by another process running in parallel[1]. The restriction form $(\nu\, a:T)P$ declares a new, *fresh* name a local to P. $P \mid Q$ denotes the parallel composition of two subprocesses P and Q. Finally, $!P$ stands for an infinite number of (parallel) copies of P.

The operational semantics of the π-calculus is defined in terms of two relations: a *structural equivalence* relation on process terms that allows the rearrangement of parallel compositions, replications and restrictions so that the participants in a communication can be brought into immediate proximity; and a *reduction* relation that describes the act of communication itself.

Structural Congruence is defined as the least congruence relation that is closed under the following rules:

1. $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$, $P \mid \mathbf{0} \equiv P$
2. $(\nu\, a)\mathbf{0} \equiv \mathbf{0}$, $(\nu\, a)(\nu\, b)P \equiv (\nu\, b)(\nu\, a)P$
3. $(\nu\, a)(P \mid Q) \equiv P \mid (\nu\, a)Q$ if $a \notin fn(P)$
4. $!P \equiv !P \mid P$

The one-step reduction relation $P \longrightarrow Q$ is the least relation closed under rules in Table 2.

The notation $P\{x_1 := M_1, \ldots x_k := M_k\}$ indicates the simultaneous substitution of M_i for each free occurrence of the variable x_i in P, for $i \in [1..k]$. We assume that substitution maps variables to names (or else unstructured values). In other words, the substitution $\{x_1 := M_1, \ldots x_k := M_k\}$ is only defined when each of the M_i is either a name or a basic value. In all other cases it is undefined.

The rule (COMM) is the core of reduction relation, as it defines the effect of synchronization between two processes on a channel. The rules (STRUCT) complete the definition. Notice that reduction is possible under a restriction, but not under either

[1] There exists an *asynchronous* variant of the calculus in which output is non-blocking. We will discuss it briefly below, and return on it in later sections, when discussing some of the derivative calculi.

Table 2 Reduction Relation

(COMMUNICATION)

$$n(x_1 : T_1, \ldots, x_k : T_k)P \mid \overline{n}\langle M_1, \ldots, M_k\rangle.Q \longrightarrow P\{x_1 := M_1, \ldots, x_k := M_k\} \mid Q$$

(STRUCTURAL RULES)

$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \frac{P \longrightarrow P'}{(\nu a)P \longrightarrow (\nu a)P'} \qquad \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$$

of the two input and output prefix forms. It is also instructive to comment on the last structural rule for reduction, that connects the relations of reduction and structural congruence, and specifically on the interplay between the reduction rule (COMM) and the structural rule $(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q$ if $a \notin fn(P)$, known as the rule of *scope extrusion*. If we read the equivalence from left to right there is nothing surprising: since the name a does not occur free in P, restricting a on this process is vacuous, and we may safely move the restriction to Q without changing (at least intuitively) the meaning of the term. When used from right to left, instead, the equivalence enables the communication of private names. Consider the term $c(x).P \mid (\nu a)\overline{c}\langle a\rangle.Q$. In their current form, the two parallel processes may not communicate. However, we may use the congruence rules to rearrange the term as in $(\nu a)(c(x).P \mid \overline{c}\langle a\rangle.Q)$, and then use (COMM) to reduce it to $P\{x := a\} \mid Q$. By effect of the reduction, the name a, which was private to Q, has now been communicated to P. Interestingly, the name a may very well be the name of a channel, which implies that the reduction has the effect of establishing a new communication link between the two processes P and Q. Also note that the new link is now *private* to P and Q, and will remain so as long as the two processes do not communicate it to third parties.

This simple example shows that the combination of scope extrusion and communication provides a very powerful mechanism for:

- dynamically changing the topological structure of a system of processes, by creating new, fresh, communication links.
- establishing *private*, hence *secure* communication links among the principals of the system.

The ability to represent dynamically changing system topologies is the distinctive feature of the π-calculus with respect to previous CCS-like calculi for concurrency. The possibility of establishing private channels, in turn, makes the π-calculus a good foundation for studying formal models of security protocols. We briefly illustrate this potential of the π-calculus with a simplified version of the protocol known as the *Wide Mouthed Frog* protocol. In this version, we have two principals A and B (the outfamous Alice and Bob), willing to exchange secret data M, and a server S, that mediates their communication:

Message 1: $A \rightarrow S$ c_{AB} on c_{AS}

Message 2: $S \rightarrow B$ c_{AB} on c_{BS}

Message 3: $A \rightarrow B$ M on c_{AB}

Initially, each one of A and B shares a channel with S. A sends to S a secret channel that it wishes to use for communicate with B; S sends this channel to B and then A and B may communicate. The π-calculus formulation of the protocol is just as direct, but now formal:

$$A \triangleq (\nu\, c_{AB})\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle M\rangle$$
$$S \triangleq c_{AS}(x).\overline{c_{BS}}\langle x\rangle$$
$$B \triangleq c_{BS}(x).x(y).P\{y\}$$

The notation $P\{y\}$ is used here simply to emphasize that P will do something with the message it receives on the input channel x. The example shows how a secret channel may be established for communication, and relies critically on scope extrusion: the scoping rules guarantee that the context in which the protocol is executed (i.e. any process running in parallel with A, B and S) will not be able to access the secret channel c_{AB}, unless of course any of the principals involved in the protocol gives it away.

This use of private channels for secrecy is suggestive and effective in its simplicity. On the other hand, a problem with the π-calculus formulation of the protocol arises when we consider its implementation in a distributed environment. In that case, it is not realistic to rely only on the scope rules to ensure secrecy of names, as one also needs to prevent the context from having free access public channels over which private names are communicated. In our example, the name c_{AB} is secret, but to guarantee that secrecy is preserved through the protocol we should also envisage a mechanism for preventing the context from reading the name c_{AB} while it is communicated over the public channels c_{AS} and c_{BS}. Unfortunately, the π-calculus does not allow one to express the cryptographic operations that would typically be used for that purpose. This observation motivated the design of the cryptographic extension of the π-calculus known as the *spi* calculus [5, 10].

We conclude the description of the untyped π-calculus with a more complex example that illustrates the reduction semantics and the computational flavor of the calculus.

Example 1 (Memory Cells). A memory cell can abstractly be thought of as an object with private store s holding the cell value, and two methods *get* and *put* for reading and writing the contents of the cell. In the π-calculus, this can be represented as a process consisting of three parallel subprocesses like the ones displayed below:

$$\begin{aligned}\mathsf{cell}(n) ::= (\nu\, s)(&\overline{s}\langle n\rangle\\ &\mid !\,get(y).s(x).(\overline{s}\langle x\rangle \mid \overline{y}\langle x\rangle)\\ &\mid !\,put(y,v).s(x).(\overline{s}\langle v\rangle \mid \overline{y}\langle\rangle))\end{aligned}$$

$\mathsf{cell}(n)$ declares the private name s representing the physical location holding the value n, and provides the two handlers for serving the "get" and "put" requests on its contents. Both the handlers are implemented as replicated processes, to make it possible to serve multiple requests. Each request is served by first spawning a fresh copy of the corresponding handler by means of the congruence rule $!P \equiv P \mid !P$.

The intuition is as follows. To read the cell contents, a user sends a "get" request by transmitting, over the channel *get*, the name of a channel where it waits for the result of the request. Upon receiving the channel name, the "get" handler inside the cell consumes the current cell value, and then reinstates it while also copying it to the

channel it received from the user. The protocol for serving a "put" request is similar. The cell's "put" handler waits for a for value v: once v is received, the handler consumes the current cell value and writes v on the private channel s. There is a further subtlety, however, in that the "put" handler inside the cell also expects an "ack" channel from the user, which it uses to signal the completion of the protocol to the user. This may be required by a user that, say, increments the cell value and then reads the new value to print it: before reading the value, the user may use the ack channel to make sure it prints the new cell value, the one resulting from the increment.

Here, we illustrate the reduction semantics with a simpler (and less realistic) user:

$$\mathsf{user}(v) ::= (\nu\, ack)(\overline{put}\langle ack, v\rangle.ack().(\nu\, ret)\overline{get}\langle ret\rangle.ret(x).\overline{print}\langle x\rangle)$$

The user first writes a new value and then reads the cell contents to print the returned value. Now consider the system $\mathsf{cell}(0) \mid \mathsf{user}(v)$. An initial phase of structural rearrangements brings the system in the form $(\nu s)(\nu\, ack)(\nu\, ret)(\ldots)_{cell} \mid (\ldots)_{user}$. Then the system $(\ldots)_{cell} \mid (\ldots)_{user}$ evolves as follows: we omit the application of congruence rules and, at each reduction step, we only display the subterms that are relevant to the reduction in question:

$$\begin{aligned}
&(\overline{s}\langle 0\rangle \mid (\mathbf{put}(\mathbf{y},\mathbf{v}).s(x).\ldots \mid \ldots))_{cell} \mid (\overline{\mathbf{put}}\langle \mathbf{ack}, \mathbf{1}\rangle.\ldots)_{user}\\
&\longrightarrow (\overline{\mathbf{s}}\langle \mathbf{0}\rangle \mid \mathbf{s}(\mathbf{x}).(\overline{s}\langle 1\rangle \mid \overline{ack}\langle\rangle) \mid \ldots))_{cell} \mid (ack().\ldots)_{user}\\
&\longrightarrow (\overline{s}\langle 1\rangle \mid \overline{\mathbf{ack}}\langle\rangle \mid \ldots)_{cell} \mid (\mathbf{ack}().ret(x).\ldots)_{user}\\
&\longrightarrow (\overline{s}\langle 1\rangle \mid (\mathbf{get}(\mathbf{y}).s(x).\ldots))_{cell} \mid (\overline{\mathbf{get}}\langle \mathbf{ret}\rangle.\ldots)_{user}\\
&\longrightarrow (\overline{\mathbf{s}}\langle \mathbf{1}\rangle \mid (\mathbf{s}(\mathbf{x}).(\overline{s}\langle x\rangle \mid \overline{ret}\langle x\rangle)\ldots))_{cell} \mid ret(x).\overline{print}\langle x\rangle\\
&\longrightarrow (\overline{s}\langle 1\rangle \mid \overline{\mathbf{ret}}\langle \mathbf{1}\rangle\ldots)_{cell} \mid \mathbf{ret}(\mathbf{x}).\overline{print}\langle x\rangle\\
&\longrightarrow (\overline{s}\langle 1\rangle \mid \ldots)_{cell} \mid \overline{print}\langle 1\rangle
\end{aligned}$$

□

1.2 Further Reading

Starting with the original presentation [46], there is by now an extensive literature on the π-calculus, also in the form of introductory [45], and advanced [54]. Most versions of the π-calculus, including the one we have outlined here, are first-order in that they allow only names to be transmitted over channels. Higher-order versions of the calculus have been extensively studied by Sangiorgi [54].

2 Typing and Subtyping for the Pi Calculus

We have so far ignored the typing annotations occurring in the input and restriction binders. Now we take them more seriously, and look at the rôle of types in the calculus. There are in fact several reasons why types are useful for process calculi in general, and for the π-calculus in particular.

- The theory of the pure (untyped) π-calculus is often insufficient to prove some "expected" properties of processes. These properties arise typically from the programmer using names according to some intended principle or logical discipline which, however, does not appear anywhere in the terms of the pure π-calculus, and therefore cannot be used in proofs. Types bring the intended structure back into light, and therefore enhance formal reasoning on process terms: for instance, typed behavioral equivalences are easier to prove, based on the fact that only typed contexts need to be considered.
- types may be employed to ensure that process interaction happens only in type-consistent ways, and hence to enable static detection of run-time type errors. To exemplify, consider the following two terms:

$$\overline{a}\langle b,c\rangle.P \mid a(x).Q \qquad \overline{a}\langle true\rangle.P \mid a(x).x(y).Q$$

  Both terms are, at least intuitively, ill-formed. The first reduces to the non-sensical process $\langle b,c\rangle(x).Q$, while the second to the ill-formed term $true(y).Q$ A simple arity check would be enough to rule out the first term as ill-formed. This, however, is not true of the second term.
- types can be useful for resource control. In the π-calculus, resources are channels, and the way that resources can be protected from unintended use is by *hiding* their names by means of the restriction operator. However, this is often too coarse a policy to enable effective resources control. In the untyped calculus, resource protection is lost when the resource name is transmitted, as no assumption can be made on how the recipient of the name will use the resource. Types may come to the rescue, as they can be employed to express and enforce a restrictive use of channels by associating them with *read* and/or *write* capabilities.

The study of type systems for process calculi originated from ideas by Milner [42, 43], based on the observation that channels used in system of processes naturally obey a discipline in the values they carry, that reflects their intended use. For instance, in the cell example above, the *ret* channel is used to communicate integers, while the *get* channel is used to communicate another channel (in fact, the *ret* channel). In Milner's original formulation, the cell example could be described by the following *sorting*:

$$\begin{array}{ll} ret : S_i & S_i \mapsto \mathsf{int} \\ get : S_g & S_g \mapsto (S_i) \\ ack : S_a & S_a \mapsto () \\ put : S_p & S_p \mapsto (S_a,()) \end{array}$$

The key idea, in types systems for the π-calculus, is that sorts, or types, are assigned only to channels, whereas processes are either well typed under a particular set of assumptions for their bound and free names and variables, or they are not. As we shall see, a different approach is possible, based on assigning more informative types to processes to describe various forms of process behavior. For the time being, however, we look at typing systems where the rôle of types is essentially that of describing (and prescribing) the intended use of channels.

The foundational work on type system by Pierce and Sangiorgi [47] was inspired by Milner's initial idea, which they elaborate in two dimensions. First they replace matching of types "by-name" with a more direct notion of structural matching, a technical modification that enables a substantively more concise and elegant presentation. Secondly, and more importantly, they employ types in a prescriptive manner to control and restrict access to channels. Their technique is based on associating channels with capabilities, and on introducing a notion of subtyping to gain additional control over the use processes can make of channels. The rest of this section gives an overview of their work. The reader is referred to [47] for full details.

2.1 Types

The structure of types is described by the following productions.

Types S,T	::=	B	types of basic values
	$\mid$	$(T_1,\ldots,T_k)$	tuple, $k \geqslant 0$
	$\mid$	$\mathsf{r}(T)$	input channel
	$\mid$	$\mathsf{w}(T)$	output channel
	$\mid$	$\mathsf{rw}(T)$	input/output channel

The type of a channel not only describes the type T of the values it carries, but also the kind of access the channel offers to its users. In the untyped calculus every channel is available for input and output: types help distinguishing, and restricting, the use of channels by associating them with access capabilities, providing users with the right to read from and/or write to a channel. The distinction between the two forms of access is reminiscent of a corresponding distinction that is made for the *reference types* in some functional programming languages. Reference types, that is, the types of mutable cells, are modeled with two different types: one for use of cells as "sources" of values, from which values can be read, and the other for cells as "sinks" where values can be placed. The same intuition applies to channels: channels of type $\mathsf{r}(T)$ may only be used as sources (i.e. for input), channels of type $\mathsf{w}(T)$ as sinks (i.e. for output), whereas channels of type $\mathsf{rw}(T)$ are input-output channels behaving both as sources and sinks.

To exemplify, $\mathsf{r}(\mathsf{int})$ is a read-only channel carrying values of type int. Since channels themselves are values, one can define a typed channel $c : \mathsf{rw}(\mathsf{r}(\mathsf{int}))$, conferring c the capability of sending and receiving values which in turn are read-only channels carrying integers.

2.2 Typing Rules

The typing rules are given in Table 3. They derive judgments in two forms: $\Gamma \vdash M : T$ stating that term M has type T, and $\Gamma \vdash P$ which simply says the process P is well-typed in *context*, or *type environment* Γ. A type environment Γ contains a set of type assumptions for the free names and variables occurring in P: equivalently, one may think of Γ as a finite map from names and variables to types.

The rules (BASE) and (TUPLE) should be self-explained. The (NAME) rule depends on the subtype relation $S \leqslant T$ which we discuss below: if the name (or variable) u is assumed to have type S in Γ, then any occurrence of that name in a process may also be

Table 3 Typing Rules for the Pi Calculus

Typing of Terms

$$\text{(BASE)}\ \frac{}{\Gamma \vdash bv : \mathsf{B}} \qquad \text{(NAME)}\ \frac{\Gamma(u) = S \quad S \leqslant T}{\Gamma \vdash u : T} \qquad \text{(TUPLE)}\ \frac{\Gamma \vdash M_i : T_i \quad i \in [1..k]}{\Gamma \vdash (M_1, \ldots, M_k) : (T_1, \ldots, T_k)}$$

Typing of Processes

$$\text{(INPUT)}\ \frac{\Gamma \vdash u : \mathsf{r}(\tilde{T}) \quad \Gamma, \tilde{x} : \tilde{T} \vdash P \quad \tilde{x} \cap \mathsf{Dom}(\Gamma) = \varnothing}{\Gamma \vdash u(\tilde{x} : \tilde{T}).P} \qquad \text{(OUTPUT)}\ \frac{\Gamma \vdash u : \mathsf{w}(T) \quad \Gamma \vdash M : T \quad \Gamma \vdash P}{\Gamma \vdash \overline{u}\langle M \rangle.P}$$

$$\text{(DEAD)}\ \frac{}{\Gamma \vdash \mathbf{0}} \qquad \text{(PAR)}\ \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \text{(REPL)}\ \frac{\Gamma \vdash P}{\Gamma \vdash\, !P} \qquad \text{(RESTR)}\ \frac{\Gamma, a : T \vdash P \quad a \notin \mathsf{Dom}(\Gamma)}{\Gamma \vdash (\nu a : T)P}$$

Table 4 Core Subtype rules for channel types

$$\text{(SUB INPUT)}\ \frac{S \leqslant T}{\mathsf{r}(S) \leqslant \mathsf{r}(T)} \qquad \text{(SUB OUTPUT)}\ \frac{T \leqslant S}{\mathsf{w}(S) \leqslant \mathsf{w}(T)} \qquad \text{(SUB IO/I)}\ \frac{}{\mathsf{rw}(T) \leqslant \mathsf{r}(T)} \qquad \text{(SUB IO/O)}\ \frac{}{\mathsf{rw}(T) \leqslant \mathsf{w}(T)}$$

typed at T provided that T is a super-type of S. The (INPUT) and (OUTPUT) rules ensure that channels are used consistently with their types. In the (INPUT) rule, the first premise requires that the channel from which input is requested provide a read capability and that the type of the input variables of the channel be consistent with the channel type. In addition, in order for the process $u(\tilde{x} : \tilde{T}).P$ to be well typed, the continuation P must also be well typed under the additional assumptions that the input variables $\tilde{x}$ are of the declared types. The rule (OUTPUT) has a similar reading. The remaining rules are easily explained: (PAR) and (REPL) are purely structural, (DEAD) states that the inert process is well typed, and (RESTR) is standard.

2.3 Subtyping

The subtype relation is central to the use of the type system to enforce access control over channels. The core subtyping rules are defined in Table 4. The subtype relation is the least reflexive and transitive relation that is closed under these rules, and a rule that extends the subtype relation homomorphically to tuples: $(S_1, \ldots, S_k) \leqslant (T_1, \ldots, T_k)$ if $S_i \leqslant T_i$ for all $i \in [1..k]$.

The two rules (SUB INPUT) and (SUB OUTPUT) are readily understood by analogy between channel types and reference types. Alternatively, one may think of a channel as a function: in its role as a source the channel returns a value, in its role as a sink it

receives argument. Now, the two rules reflect the subtype relation for function types: covariant in their return type and contra-variant in their input. The rules (SUB IO/I) and (SUB IO/O) enable access control: any channel (which in the untyped calculus is always available for both input and output) may be associated with a more restrictive type (read-only or write-only) to protect it from misuse in certain situations. To illustrate the power of subtyping for resource access control, consider the following example from [47].

Example 2 (Access to a Printer). Suppose we have a system with a printer P and two clients C_1 and C_2. The printer provides a request channel p carrying values of some type T representing data to be printed on behalf of the clients. The system can be represented by the π-calculus process $(\nu\, p : \mathsf{rw}(T))(P \mid C_1 \mid C_2)$.

If we take, say, $C_1 \triangleq \overline{p}\langle j_1\rangle.\overline{p}\langle j_2\rangle.\ldots$, one would expect that the jobs $j_1, j_2, \ldots$ are received and processed, in that order, by the printer P. This is not necessarily the case, however, as C_2 might be not be willing to comply with the rules of the protocol. For instance, it competes with P to "steal" the jobs sent by C_1 and throws them away: $C_2 \triangleq !\, p(j : T).\mathbf{0}$.

One can prevent this kind of misbehavior by constraining the capabilities offered to C_1 and C_2 on the channel p: in the end, the clients should only write on p, whereas the printer should only read from it. We may therefore extend the system with an initialization phase that enforces this intended behavior on all the participants in the protocol. The initialization phase uses two channels, a and b, to communicate the name p to the printer and to the two clients, restricting the respective capabilities on p.

$$(\nu\, p : \mathsf{rw}(T))\ (\overline{a}\langle p\rangle.\overline{b}\langle p\rangle \mid a(x : \mathsf{r}(T)).P \mid b(y : \mathsf{w}(T)).(C_1 \mid C_2))$$

Notice that now p is a read-only channel within P and a write-only channel within C_1 and C_2. Assuming appropriate definitions for the processes P, C_1 and C_2, the system type checks, under the assumption $a, b : \mathsf{rw}(\mathsf{rw}(T))$, as the subtype relation ensures that $p : \mathsf{rw}(T)$ may legally be substituted for any $x : \mathsf{r}(T)$ or $y : \mathsf{w}(T)$.

2.4 Properties of the Type System

The type system satisfies the standard properties one expects: subject reduction and type safety. In functional languages, subject reduction guarantees that types are preserved during the computation. The result for the π-calculus is similar, and ensures that well-typedness is preserved by all the non-deterministic reductions of a process.

Theorem 1 (Subject Reduction). *If* $\Gamma \vdash P$ *and* $P \longrightarrow Q$, *then* $\Gamma \vdash Q$.

The proof of this result requires two auxiliary results. The first is the so-called subject-congruence theorem, stating that well-typedness is preserved by the relation of structural congruence.

Theorem 2 (Subject Congruence). *If* $\Gamma \vdash P$ *and* $P \equiv Q$, *then* $\Gamma \vdash Q$. *Dually, if* $\Gamma \vdash P$ *and* $Q \equiv P$, *then* $\Gamma \vdash Q$.

The second is the π-calculus version of the familiar substitution lemma from type system for the λ-calculus.

Theorem 3 (Substitution). *If* $\Gamma \vdash u : T$ *and* $\Gamma, x : T \vdash P$, *then* $\Gamma \vdash P\{x := u\}$.

Type safety is a more subtle issue. In functional calculi, proving type safety amounts to proving the so-called *absence of stuck states*, i.e. to show that the evaluation of well-typed terms either does not terminate, or returns a *value*, for a suitable notion of value. In the π-calculus, there is no notion of value, as computation is entirely based on interaction between processes, that do not return values. A notion of "stuck state" may nevertheless be formulated, and taken as the basic common denominator to different notions of type safety.

Theorem 4 (Basic Type Safety). *Assume* $\Gamma \vdash P$, *and* $P \longrightarrow Q$. *If* Q *contains a subterm*

$$c(x_1 : T_1, \ldots, x_n : T_n).Q_1 \mid \overline{c}\langle M_1, \ldots, M_k \rangle.Q_2$$

then all of the following hold true: c *is a name or variable (i.e. not a constant of basic type),* $k = n$ *and each of the* M_i *is a non-structured value.*

The theorem says essentially that process interaction happens in type-consistent ways, and never generates undefined substitutions. In addition, one may wish to prove other properties for reduction, and consequently richer notions of type safety. For instance, for the type system we have presented in the previous section, it can be proved that reduction of well-typed processes preserves guarantees that access to channels by processes is always consistent with the capabilities conferred to the channels by their types. We will discuss type-safety more formally in some of the calculi presented in later sections. Presently, we content ourselves with this informal formulation, and refer the interested reader to [47] for details on this richer notion of type safety.

2.5 Further Reading

The study of type systems for the π-calculus is currently very active, and has produced a large body of literature. Besides the work by Pierce and Sangiorgi we have reviewed in this section, and those we will discuss later on, an interesting pointer is to the work of Igarashi and Kobayashi [37] where a generic framework is proposed in which to understand several previous systems.

3 The Join Calculus

The *Join calculus* [29, 30] is a variant of the asynchronous π-calculus [12, 36] which combines restriction, reception, and replication in one construct, the *join receptor*: $J \triangleright P$. For example the definition

$$\texttt{def apply}\langle \texttt{f}, \texttt{x} \rangle \triangleright \texttt{f}\langle \texttt{x} \rangle \tag{1}$$

defines a new name `apply` that receives two arguments and apply the first to the second. More precisely it receives a channel name that it bounds to f and a name that it bounds to x and sends the latter over the former. This is more formally shown by the following reduction:

$$\texttt{def apply}\langle \texttt{f}, \texttt{x} \rangle \triangleright \texttt{f}\langle \texttt{x} \rangle \texttt{ in apply}\langle \texttt{g}, \texttt{y} \rangle \quad \longrightarrow \quad \texttt{def apply}\langle \texttt{f}, \texttt{x} \rangle \triangleright \texttt{f}\langle \texttt{x} \rangle \texttt{ in g}\langle \texttt{y} \rangle$$

Table 5 The Join calculus

Processes	P,Q	$::= x\langle\widetilde{V}\rangle$	Asynchronous message on x
		$\texttt{def}\ D\ \texttt{in}\ P$	Definition of D in P
		$P \mid Q$	Parallel Composition
		$\mathbf{0}$	Empty Process
Join patterns	J,J'	$::= x\langle\tilde{y}\rangle$	Asynchronous reception on x
		$J \mid J'$	Joining messages
Definition	D,E	$::= J \triangleright P$	Elementary clause
		$D \wedge E$	Simultaneous definition
Values	V,V'	$::= \tilde{x}$	Names

Table 6 Received, defined and free variables

$dv(J \triangleright P)$	$= dv(J)$	$dv(D \wedge E)$	$= dv(D) \cup dv(E)$
$dv(T)$	$= \emptyset$	$dv(J \mid J')$	$= dv(J) \cup dv(J')$
$dv(x\langle\tilde{v}\rangle)$	$= \{x\}$		
$rv(x\langle\tilde{v}\rangle)$	$= \{u \mid u \in \tilde{v}\}$	$rv(J \mid J')$	$= rv(J) \uplus rv(J')$
$fv(J \triangleright P)$	$= dv(J) \cup (fv(P) - rv(J))$	$fv(D \wedge E)$	$= fv(D) \cup fv(E)$
$fv(\varepsilon)$	$= \emptyset$		
$fv(x\langle\tilde{v}\rangle)$	$= \{x\} \cup \{u \in \tilde{v}\}$		
$fv(\texttt{def}\ D\ \texttt{in}\ P)$	$= (fv(P) \cup fv(D)) - dv(D)$		
$fv(P \mid Q)$	$= fv(P) \cup fv(Q)$		
$fv(0)$	$= \emptyset$		

The syntax of the calculus is given in Table 5, where we assume names $x, y, \ldots$ to be drafted from an infinite set $\mathcal{N}$.

The only binding mechanism is the join pattern: the formal parameters which are received are bound in the guarded process. The *received* variables, $rv(J)$, are the names to which the messages sent are bound; the *defined* variables in a join pattern or a definition, $dv(J)$ and $dv(D)$, are the names which are bound by the definition. The *free* variables, $fv(P)$ and $fv(D)$, are all the names which are not bound. Received, defined and free variables can be easily defined as expected by structural induction (see Table 6).

It is important to notice that there is no linearity condition on the channel names in a composed join pattern: however, elementary join patterns are required to be linear, i.e. received variables are supposed to be pairwise distinct. A name is said to be *fresh* in a process when it is not free in it. In the following discussions a consistent use of names is assumed.

The operational semantics of the Join calculus is given using the chemical paradigm (structural rules $\rightleftharpoons$ plus reduction $\rightarrow$) in terms of the so called Reflexive Chemical Ab-

Table 7 The RCHAM

(str-join)	$\models P \mid Q$	$\rightleftharpoons$	$\models P, Q$
(str-def)	$\models \texttt{def}\ D\ \texttt{in}\ P$	$\rightleftharpoons$	$D_{\sigma_{dv}} \models P_{\sigma_{dv}}$
(red)	$\ldots \wedge J \triangleright P \wedge \ldots \models J_{\sigma_{rv}}$	$\rightarrow$	$\ldots \wedge J \triangleright P \wedge \ldots \models P_{\sigma_{rv}}$

Side conditions: in *(str-def)* σ_{dv} instantiates the names in $dv(D)$ to distinct fresh names; in *(red)* σ_{rv} substitutes the received variables $rv(J)$ with the values actually received

stract Machine (RCHAM) [29, 14] (see Table 7). States of the RCHAM are expression of the form $D \models P$, where P are the running processes and D are the (chemical) reactions.

Note that join patterns can be the parallel composition of different receptions, and that reduction takes place only when all the receptions synchronize. So for example the following receptor

$$\texttt{def ready}\langle\texttt{printer}\rangle \mid \texttt{print}\langle\texttt{file}\rangle \triangleright \texttt{printer}\langle\texttt{file}\rangle\ \texttt{in}\ P$$

reduces only when in P two (unbound) outputs on `ready` and `print` occur in parallel as for

$$\texttt{def ready}\langle\texttt{printer}\rangle \mid \texttt{print}\langle\texttt{file}\rangle \triangleright \texttt{printer}\langle\texttt{file}\rangle\ \texttt{in ready}\langle\texttt{gutenberg}\rangle \mid \texttt{print}\langle\texttt{myths.ps}\rangle \mid Q$$

which reduces to

$$\texttt{def ready}\langle\texttt{printer}\rangle \mid \texttt{print}\langle\texttt{file}\rangle \triangleright \texttt{printer}\langle\texttt{file}\rangle\ \texttt{in gutenberg}\langle\texttt{myths.ps}\rangle \mid Q$$

The same behavior could be obtained by composing this definition with the definition (1):

$$\texttt{def apply}\langle\texttt{f},\texttt{x}\rangle \triangleright \texttt{f}\langle\texttt{x}\rangle \wedge \texttt{ready}\langle p\rangle \mid \texttt{print}\langle f\rangle \triangleright \texttt{apply}\langle p, f\rangle$$

3.1 Typing

Let us again consider the definition of the expression (1). If we use $\langle T\rangle$ to denote the type of channels transporting values of type T, then `apply` has type $\langle\langle T\rangle, T\rangle$ for every type T. In words `apply` is a channel that transports pairs formed by a channel and a value that can be sent over that channel.

Note the polymorphic nature of the type of `apply`. This can be formally expressed by generalizing the type of `apply` into the following type schema: $\forall\alpha.\langle\langle\alpha\rangle, \alpha\rangle$. We saw before that join calculus provides synchronization between join patterns. Thus for instance a variant of `apply` that receives `f` and `x` from different sources can be defined as follows

$$\texttt{def fun}\langle\texttt{f}\rangle \mid \texttt{arg}\langle\texttt{x}\rangle \triangleright \texttt{f}\langle\texttt{x}\rangle$$

Table 8 Typing rules for the Join Calculus

$$\text{(INST)}\ \frac{x : \forall\tilde{\alpha}.T \in A}{A \vdash x : T\{\tilde{\alpha} := \tilde{T}'\}} \qquad \text{(PAR)}\ \frac{A \vdash P \quad A \vdash Q}{A \vdash P \mid Q}$$

$$\text{(MESSAGE)}\ \frac{A \vdash x : \langle T_1, \ldots, T_n\rangle \quad A \vdash v_i : T_i \quad (i = 1..n)}{A \vdash x\langle v_1, \ldots, v_n\rangle} \qquad \text{(DEF)}\ \frac{A, B \vdash D :: B \quad A, Gen(B,A) \vdash P}{A \vdash \texttt{def}\ D\ \texttt{in}\ P}$$

$$\text{(JOIN)}\ \frac{A, y_{ij} : T_{ij}^{i=1..n, j=1..m_i} \vdash P}{A \vdash x_1\langle y_{1j}^{j=1..m_1}\rangle \mid \ldots \mid x_n\langle y_{nj}^{j=1..m_n}\rangle \triangleright P :: x_i : \langle T_{i1}, \ldots T_{im_i}\rangle^{i=1..n}}$$

$$\text{(AND)}\ \frac{A \vdash D_1 :: B_1 \quad A \vdash D_2 :: B_2}{A \vdash D_1 \wedge D_2 :: B_1, B_2}\ (B_1|_{\mathsf{Dom}(B_2)} = B_2|_{\mathsf{Dom}(B_1)})$$

According to what we said before `fun` and `arg` can be respectively typed as $\langle\langle\alpha\rangle\rangle$ and $\langle\alpha\rangle$. Observe, however, that `fun` and `arg` are correlated in their types as they must share the same type variable α. This forbids to generalize their types separately: if we assigned them the types $\forall\alpha.\langle\langle\alpha\rangle\rangle$ and $\forall\alpha.\langle\alpha\rangle$, then the correlation of the types of the two names defined in the same join pattern would be lost. In [14] this problem is handled by the definition of the generalization rule that forbids the generalization of type variables that appear free in the type of more than one co-defined name.

The type system of [14] is defined as follows:

$$\textit{Types}\quad T ::= \alpha \mid \langle T, \ldots, T\rangle \qquad \textit{Type Envs}\quad B ::= \varnothing \mid B, x : T$$

$$\textit{Schemas}\ \sigma ::= T \mid \forall\alpha.\sigma \qquad \textit{Schema Envs}\ A ::= \varnothing \mid A, x : \sigma$$

The type system includes three kinds of typing judgments:

$A \vdash u{:}T$ the name u has type T in A
$A \vdash P$ the process P is well typed in A
$A \vdash D :: B$ the definition D is well-typed in A with types B for its defined types

which are deduced by the typing rules in Table 8.

In that table, $Gen(B,A)$ is the generalization of the type environment B of the form $(x_i : T_i)^{i=1..n}$ with respect to the schema environment A: let $fv(A)$ be the set $\cup_{(s:\sigma)\in A} vars(\sigma)$ with $vars(\sigma)$ is the set of variables occurring in σ; let $B \setminus x$ be the environment B without the binding for x; then $Gen(B,A)$ is $(x_i : \forall(fv(T_i) - fv(A, (B \setminus x_i))).T_i)^{i=1..n}$.

With the exception of (DEF) all the rules are straightforward, insofar as they are almost directly inspired by the typing rules for polymorphic (poliadic) λ-calculus. (INST) assigns to a variable x any type that is an instance of the type schema associated to the x in A; (PAR) is straightforward; (MESSAGE) checks that the types of the actual parameters match the type of the channel the parameters are sent over; (JOIN) checks that the guarded process P is typable under the assumption that the types of the formal parameters of the join patterns match those of the corresponding channels, and associates to the definition the type environment of its declared names; (AND) associates to the composition of two definitions the composition of the type environments of their declared names, provided that the two definitions do not declare a common name. Finally (DEF) is the most technical rule: first it checks the typing of the definition D under the type environment produced by D. This allows recursive definitions; second it checks the well typing of P under the generalization of the types of the new definition with respect to A. In particular the generalization takes into account the problem of sharing we hinted in the beginning of the section. Therefore for every constraint $x{:}T \in B$ the generalization does not generalize all the free type variables of T but, instead, only those free variables that are not shared with a previous definition or with a parameter of the actual join pattern.

3.2 Properties of the Type System

Soundness. The soundness of the type system is obtained by proving subject reduction and basic type safety (corresponding to Theorem 4 for π-calculus.)

Theorem 5 (Subject Reduction). *If $A \vdash P$ and $P \longrightarrow Q$, then $A \vdash Q$.*

Definition 1. *A process of the form* $\texttt{def}\ D \wedge J \triangleright\ \texttt{in}\ Q \mid x\langle\tilde{v}\rangle$ *is wrong if J contains a message $x\langle\tilde{y}\rangle$ where $\tilde{y}$ and $\tilde{v}$ have different arities.*

Theorem 6 (Basic Type Safety). *If $A \vdash P$ then P is not wrong.*

The composition of the previous two theorems ensures that well typed processes never go wrong.

Type Inference. Finally, there exists an algorithm that for every typable process returns the most general schema environment under which the process can be typed, while it fails if it is applied to a process that is not typable.

3.3 Further Reading

In [7] Abadi, Fournet, and Gonthier define the sjoin-calculus, that extends the join calculus with constructs for encryption and decryption and with names that can be used as keys, nonces, or other tags. This extension is very reminiscent of the the way the spi-calculus (see section 7) extends the π-calculus: as a matter of fact, the name sjoin was chosen in analogy with spi. The authors also show how to translate sjoin into a lower-level language that includes cryptographic primitives mapping communication on secure channels into encrypted communication on public channels. A correctness

theorem for the translation ensures that one can reason about programs in sjoin without mentioning the cryptographic protocols used to implement them in the lower-level implementation.

In [17] Conchon and Pottier advocate that the the type system of [14], that forbids the generalization of any type variable that is shared between two jointly defined names (such as `fun` and `arg`), is overly restrictive when one wants to use types in a descriptive – rather then prescriptive – way. To that end they switch from the system of [14] in which the generalization is performed on syntactic criteria, to a richer type system based on constraints and where the generalization is more "semantic" (polymorphic types are interpreted as particular sets of monomorphic types) and fairly natural. However, rather surprisingly, the new generalization criterion hinders type inference as it results very difficult (perhaps impossible) to infer a most general type. As a result they propose a more restrictive (and syntactic) criterion that, while it allows type inference, it is closer to the original system of [14].

In his PhD. thesis [15] Conchon extends the type system of JOIN(X) with information-flow annotations that ensure a noninterference property based on bisimilarity equivalences. The new systems thus obtained can detect, for instance, information flow caused by contentions on distributed resources, which are not detected, in a satisfactory way, when using testing equivalences. The achievement is however limited by the fact that equivalences, rather than congruences, are considered.

A more in depth study of bisimulation for the join calculus can be found in [11].

In all these variants, join remains a concurrent calculus. In [31] the authors define the Distributed Join Calculus that extends join calculus essential with locations, migration, and failure. The new calculus allows one to express mobile agents roaming on the net, that is, that autonomously move from some node to a different node where they resume their current execution. Distributed join is also the core of the distributed language *jocaml*[16].

4 The Pi Calculus with Groups

In Section 1 (and we will see it also in Section 7) we discussed the importance of scope extrusion for secrecy. However, inattentive use of scope extrusion may cause secrets to be leaked. Consider a process P that wants to create a private name x. In the pi-calculus this can be done by letting P evolve into a configuration $(\nu x)P'$, where the channel x is intended to remain private to P'. This privacy policy is going to be violated if the system then evolves into a situation such as the following, where p is a public channel known to an hostile process (opponent) running in parallel with P.

$$p(y).O \mid (\nu x)(\overline{p}\langle x\rangle \mid P') \tag{2}$$

In this situation, the name x is about to be sent by P over the public channel p and received by the opponent. In order for this communication to happen, the rules of the pi-calculus, described in Section 1, require first an enlargement of the scope of x. After extrusion we have:

$$(\nu x)(p(y).O \mid \overline{p}\langle x\rangle \mid P') \tag{3}$$

Now, x can be communicated over p, and the opponent acquires the secret.

The private name x has been leaked to the opponent by a combination of two mechanisms: the output instruction $\overline{p}\langle x\rangle$ and the extrusion of (νx). It seems that we need to restrict either communication or extrusion. Since names are dynamic data in the pi-calculus, it is not easy to say that a situation such as $\overline{p}\langle x\rangle$ (sending x on a channel known to the opponent) should not arise, because p may be dynamically obtained from some other channel, and may not occur at all in the code of P.

The other possibility is to prevent extrusion, which is a necessary step when leaking names outside their initial scope. However, extrusion is a fundamental mechanism in the pi-calculus: blocking it completely would also block innocent communications over p.

A natural question is whether one could somehow declare x to be private, and have this assertion statically checked so that the privacy policy of x cannot be violated. To this end, in [13] authors add an operation of group creation to the typed pi-calculus, where a group is a type for channels. Group creation is a natural extension of the sort-based type systems developed for the pi-calculus (see Section 1). However, group creation has an interesting and subtle connection with secrecy. Creation of fresh groups has the effect of statically preventing certain communications, and can block the accidental or malicious leakage of secrets.

Intuitively, no channel belonging to a fresh group can be received by processes outside the initial scope of the group, even if those processes are untyped. Crucially, groups are not values, and cannot be communicated; otherwise, this secrecy property would fail.

Starting from the typed pi-calculus, we can classify channels into different groups (usually called sorts). We could have a group G for our private channels and write $(\nu x{:}G)P$ to declare x to be of sort G. However, if groups are global (as usually happens with sorts in standard pi-calculus type systems), they do not offer any protection because an opponent could very well mention G in an input instruction, and leakage can thus be made to typecheck:

$$p(y : G).O \mid (\nu x{:}G)(\overline{p}\langle x\rangle \mid P') \tag{4}$$

In order to guarantee secrecy, the group G itself should be secret, so that no opponent can input names of group G, and that no part of the process P can output G information on public channels.

In general we want the ability to create fresh groups on demand, and then to create fresh elements of those groups. To this end, we extend the pi-calculus with an operator, $(\nu G)P$, to dynamically create a new group G in a scope P. Although group creation is dynamic, the group information can be tracked statically to ensure that names of different groups are not confused. Moreover, dynamic group creation can be very useful: we can dynamically spawn subsystems that have their own pool of shared resources that cannot interfere with other subsystems.

Consider the following process, where $G[\,]$ is the type of a channel of group G:

$$(\nu p{:}U)\ (p(y{:}T).O \mid (\nu G)(\nu x{:}G[\,])\overline{p}\langle x\rangle) \tag{5}$$

Here an attempt is made again to send the channel x over the public channel p. Fortunately, this process cannot be typed: the type T would have to mention G, in order to receive a channel of group G, but this is impossible because G is not known in the global scope where p has been declared.

The construct (νG) has extrusion properties similar to (νx), which are needed to permit legal communications over channels unrelated to G channels, but these extrusion rules prevent G from being confused with any groups mentioned in T.

Untyped Opponents. Let us now consider the case where the opponent process is untyped or, equivalently, not well-typed. This is intended to cover the situation where an opponent can execute any instruction without being restricted by static checks such as type checking or bytecode verification. For example, the opponent could be running on a separate, untrusted, machine. Let consider again the previous process, where we remove typing information from the code of the opponent, since an opponent does not necessarily respect the typing rules. The opponent now attempts to read any message transmitted over the public channel, no matter what its type is.

$$(\nu\, p{:}U)(p(y).O \mid (\nu G)(\nu x{:}G[\,])\overline{p}\langle x\rangle) \tag{6}$$

The untyped opponent will not acquire secret information by cheating on the type of the public channel. The fact that the process P is well typed is sufficient to ensure secrecy, even in the presence of untyped opponents. This is because, in order for P to leak information over a public channel p, the output operation $\overline{p}\langle x\rangle$ must be well typed. The name x can be communicated only on channels whose type mentions G. So the output $\overline{p}\langle x\rangle$ cannot be well typed, because then the type U of p would have to mention the group G, but U is not in the scope of G.

We have thus established, informally, that a process creating a fresh group G can never communicate channels of group G to an opponent outside the initial scope of G, either because a (well typed) opponent cannot name G to receive the message, or, in any case, because a well typed process cannot use public channels to communicate G information to an (untyped) opponent. Thus, channels of group G are forever secret outside the initial scope of (νG). So, secrecy is reduced in a certain sense to scoping and typing restrictions. As we have seen, the scope of channels can be extruded too far, perhaps inadvertently, and cause leakage, while the scope of groups offers protection against accidental or malicious leakage, even though it can be extruded as well.

4.1 Syntax and Operational Semantics

We start showing the syntax of an asynchronous, polyadic, typed pi-calculus with groups and group creation. Types specify, for each channel, its group and the type of the values that can be exchanged on that channel.

$$\textit{Types} \quad T ::= G[T_1,\ldots,T_n] \qquad \text{polyadic channel in group } G$$

As usual, in a restriction $(\nu x{:}T)P$ the name x is bound in P, and in an input $x(\tilde{y}:\tilde{T}).P$, the names $y_1,\ldots,y_k$ are bound in P. In a group creation $(\nu G)P$, the group G is bound with scope P. Let $fn(P)$ be the set of free names in a process P, and let $fg(P)$,$fg(T)$ be the sets of groups free in a process P and a type T, respectively.

The operational semantics of the calculus is similar to that of the typed pi-calculus described in Section 1. Group creation is handled by the following new rules of structural equivalence and reduction:

Table 9 Typed pi-calculus with Groups

Expressions	M,N	$::= a,\ldots,p$	name
		$\mid x,\ldots,z$	variable
Processes	P,Q,R	$::= \mathbf{0}$	stop
		$\mid \overline{u}\langle\tilde{M}\rangle.P$	polyadic output
		$\mid u(\tilde{x}{:}\tilde{T}).P$	polyadic input
		$\mid (\nu G)P$	group creation
		$\mid (\nu a{:}T)P$	restriction
		$\mid P \mid P$	composition
		$\mid\ !P$	replication

(Struct GRes GRes) $(\nu G_1)(\nu G_2)P \equiv (\nu G_2)(\nu G_1)P$

(Struct GRes Res) $(\nu G)(\nu x{:}T)P \equiv (\nu x{:}T)(\nu G)P$ if $G \notin fg(T)$

(Struct GRes Par) $(\nu G)(P \mid Q) \equiv P \mid (\nu G)P$ if $G \notin fg(P)$

(Red GRes) $(\nu G)P \longrightarrow (\nu G)Q$ if $P \longrightarrow Q$

Note that rule (Struct Gres Res) is crucial: it implements a sort of "barrier" between processes knowing a group name and processes that do not know it.

4.2 The Type System

Environments declare names and groups in scope during type-checking; we define environments, Γ, by $\Gamma ::= \emptyset \mid \Gamma, G \mid \Gamma, x{:}T$. We define four typing judgments: first, $\vdash \Gamma$ means that Γ is well formed; second $\Gamma \vdash T$ means that T is well formed in Γ; third, $\Gamma \vdash x : T$ means that $x : T$ is in Γ, and that Γ is well formed; and, fourth, $\Gamma \vdash P$ means that P is well formed in the environment Γ. Typing rules are collected in Table 10.

Properties of the Type System. A consequence of our typing discipline is the ability to preserve secrets. In particular, the subject reduction property, together with the proper application of extrusion rules, has the effect of preventing certain communications that would leak secrets. For example, consider the process (4) at the beginning of this section:

$$(\nu\, p{:}U)(p(y{:}T).O \mid (\nu G)(\nu x{:}G[\,])\overline{p}\langle x\rangle)$$

In order to communicate the name x (the secret) on the public channel p, we would need to reduce the initial process to the following configuration:

$$(\nu\, p{:}U)(\nu G)(\nu x{:}G[\,])(p(y{:}T).O \mid \overline{p}\langle x\rangle)$$

If subject reduction holds then this reduced term has to be well-typed, which is true only if $p : H[T]$ for some H, and $T = G[\,]$. However, in order to get to the point of

Table 10 Typing rules for the pi-calculus with groups

$$\frac{}{\vdash \emptyset}\ (\textsc{Env Empty}) \qquad \frac{\Gamma \vdash T \quad u \notin dom(\Gamma)}{\vdash \Gamma, u:T}\ (\textsc{Env } u) \qquad \frac{\vdash \Gamma \quad G \notin dom(\Gamma)}{\vdash \Gamma, G}\ (\textsc{Env Group})$$

$$\frac{G \in dom(\Gamma) \quad \Gamma \vdash T_1 \ldots \Gamma \vdash T_n}{\Gamma \vdash G[T_1, \ldots, T_n]}\ (\textsc{Type Chan}) \qquad \frac{\vdash \Gamma', x:T, \Gamma''}{\Gamma', x:T, \Gamma'' \vdash x:T}\ (\textsc{Project}) \qquad \frac{\Gamma, G \vdash P}{\Gamma \vdash (\nu G)P}\ (\textsc{GRes})$$

$$\frac{\Gamma \vdash M : G[T_1, \ldots, T_n] \quad \Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P}{\Gamma \vdash M(x_1 : T_1, \ldots, x_n : T_n)P}\ (\textsc{Input}) \qquad \frac{\Gamma, n : T \vdash P}{\Gamma \vdash (\nu n : T)P}\ (\textsc{Res}) \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathbf{0}}\ (\textsc{Dead})$$

$$\frac{\Gamma \vdash M : G[T_1, \ldots, T_n] \quad \Gamma \vdash N_1 : T_1 \ldots \Gamma \vdash N_n : T_n}{\Gamma \vdash \overline{M}\langle N_1, \ldots, N_n \rangle}\ (\textsc{Output}) \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}\ (\textsc{Par}) \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P}\ (\textsc{Repl})$$

bringing the input operation of the opponent next to the output operation, we must have extruded the (νG) and the $(\nu x{:}G[\,])$ binders outward. The rule (Struct Gres Res), used to extrude (νG) past $p(y{:}T).O$, requires that $G \notin fg(T)$. This contradicts the requirement that $T = G[\,]$.

Proposition 1 (Subject Congruence). *If* $\Gamma \vdash P$ *and* $P \equiv Q$, *then* $\Gamma \vdash Q$.

Proposition 2 (Subject Reduction). *If* $\Gamma \vdash P$ *and* $P \longrightarrow Q$, *then* $\Gamma \vdash Q$.

The formalization of secrecy is inspired by Abadi's definition [2]: a name is kept secret from an opponent if after no series of interactions is the name transmitted to the opponent. We model the external opponent simply by the finite set of names S known to it. A complete formalization of this notion of security can be found in [13], here we only overview the main theorem and its proof. The following theorem expresses the idea that in the process $(\nu G)(\nu x{:}G[T_1, \ldots, T_n])P$, the name x of the new group G is known only within P (the scope of G) and hence is kept secret from any opponent able to communicate with the process (whether or not the opponent respects our type system). Let $erase(P)$ be the process obtained from P by erasing type annotations and new-group creations. Let S be a set of names, we say that a process P *preserves the secrecy of* x *from* S if P will never communicate the name x to an opponent initially knowing the names in S.

Theorem 7 (Secrecy). *Suppose that* $\Gamma \vdash (\nu G)(\nu x{:}T)P$ *where* $G \in fg(T)$. *Let* S *be the names occurring in* $dom(\Gamma)$. *Then the erasure* $(\nu x)erase(P)$ *of* $(\nu G)(\nu x{:}T)P$ *preserves the secrecy of the restricted name* x *from* S.

The proof of the secrecy theorem (see [13]) is based on an auxiliary type system that partitions channels into untrusted channels , with type Un and trusted ones, with type $Ch[T_1,\ldots,T_n]$, where each T_i is either a trusted or untrusted type. The type system insists that names are bound to variables with the same trust level (that is, the same type), and that no trusted name is ever transmitted on an untrusted channel. Hence an opponent knowing only untrusted channel names will never receive any trusted name.

In particular, for any group G, we can translate group-based types into the auxiliary type system as follows: any type that does not contain G free becomes Un, while a type $H[T_1,\ldots,T_n]$ that contains G free is mapped onto $Ch[\langle\!\langle T_1 \rangle\!\rangle_G,\ldots,\langle\!\langle T_n \rangle\!\rangle_G]$. This translation is proved to preserve typability. This implies that an opponent knowing only names whose type does not contain G free, will never be able to learn any name whose type contains G free. This is the key step in proving the secrecy theorem.

Finally, note that the typing rules constrain only the principals that want to protect their secrets from attackers. On the contrary, there are no restrictions on the code the attackers may run; we have in fact that any untrusted opponent may be type-checked as follows.

Lemma 1. *For all P, if $fn(P) = \{x_1,\ldots,x_n\}$ then $x_1 : Un,\ldots,x_n : Un \vdash P$.*

This is a distinctive property of the approach we discussed in this section, since it makes the type system suitable for reasoning about processes containing both trusted and untrusted subprocesses.

5 The Security Pi Calculus

The security π-calculus is an extension of the π calculus defined by Hennessy and Riely [33] to study properties of *resource access* and *information flow* control in systems with *multilevel* security. Before discussing the security π-calculus, we first give a very brief overview of the underlying models of multilevel security.

5.1 Multilevel Security

Traditional models of security are centered around notions of *subjects* and objects, with the former performing accesses on the latter by *read* and *write* (as well as *append*, *execute*, ..., etc. in certain models) operations. Multilevel security presupposes a lattice of security levels, and every subject and object in the system is assigned a level in this lattice. Based on these levels, access to objects by subjects are classified as *read-up* (resp. *read-down*) when a subject attempts to read an object of higher (resp. lower) level, and similarly for write accesses. Relying on this classification, *security policies* are defined to control access to objects by subjects and, more generally flow of information among the subjects and objects of the system.

An important class of security policies are the so-called *Mandatory Access Control* (MAC) policies, among which notable examples are *defense* security and *business* security. Defense security aims at protecting confidentiality of data by preventing flow of information from *high*, privileged, subjects to *low*, users. This is accomplished by forbidding read-up's and write-down's to objects: low-level users may not read confidential information held in high-level documents, and high-level principals may not write

Table 11 Syntax: The Security pi calculus

Expressions	M,N	$::= \ldots$	as in Table 1
Processes	P,Q,R	$::= \mathbf{0}$	stop
		$\mid \overline{u}\langle N\rangle$	asynchronous output
		$\mid u(\tilde{x}:\tilde{T}).P$	input
		$\mid (\nu a:T)P$	restriction
		$\mid P \mid P$	composition
		$\mid !P$	replication
		$\mid [P]_\sigma$	process at clearance σ

information on low-level objects that may be available to low-level users. Business security, on the other hand, centers around integrity, and a weaker form of confidentiality, and provides guarantees that low-level users have no direct access to secret high-level data, either in read or write mode.

Enforcing confidentiality and integrity often requires further constraints to prevent flow of sensitive information to non-authorized subjects arising from subtle and hidden ways of transmitting information, viz. *covert channels*: these may be established in several ways, via system-wide side effects on shared system resources. The prototypical example of covert channel is realized by means of the "file system full" exception. Suppose that a process fills the file system, and then deletes a 1-bit file: further attempts by that process to write that file will inform it of any two (high-level) users exchanging 1-bit information via the file system.

5.2 Syntax of the Security Pi-Calculus

The security π-calculus is based on the *asynchronous* variant of the π calculus. The choice of asynchronous output is motivated by security reasons, as synchronous output is more prone to covert channels and implicit flow of information. We will return to this point later: as of now, we proceed with our discussion on the asynchronous π-calculus and its extension with security.

There are different ways that the asynchronous π-calculus can be defined: for instance, one may define it by relying on the same syntax given in Table 1, and by extending the relation of structural congruence with the new rule: $a\langle M\rangle.P \equiv a\langle M\rangle.\mathbf{0} \mid P$. This rule effectively leads to an asynchronous version of the output operation, as it allows the process P to reduce, hence evolve, independently of the presence of an input process consuming the value M sent over the channel a.

Here, however, we will adhere to the more standard practice and use a different syntax in which output on a channel is defined as a process rather than a prefix. The syntax of the security π-calculus results from the syntax of the π-calculus from this change and from introducing a new construct for processes.

As anticipated, the output construct is now a process rather than a prefix: this is all that is needed to account for asynchrony. The new syntactic form $[P]_\sigma$ denotes a process P running at security level σ; it has no real computational meaning, as the notion of reduction is not affected by this construct. It is, however, relevant to the definition of

the instrumented semantics that we will introduce to capture a notion of run-time error resulting from *security violations.*

In the instrumented semantics, we view processes as playing the rôle of subjects, while channels are naturally associated with the rôle of objects that processes access in read and write mode. Security levels are associated with channels by enriching the structure of channel types: besides associating input-output capabilities with each name, channel types also include a security level.

The structure of the types is defined in terms of a lattice *SL* of security levels. We let Greek letters like $\delta, \sigma, \rho, \ldots$ range over the elements of this lattice: the top and bottom elements are denoted by $\top$ and $\bot$ as usual. To enhance flexibility, the structure of types allows different security levels to be associated with the input and output capabilities for a channel. Thus, if S and T are types, channels types may be structured as shown in the following to examples:

- $\{\mathsf{w}_\bot(S), \mathsf{r}_\top(T)\}$: the type of channels where low processes can write (values of type S), and only high processes can read (values of type T). This typing is appropriate for a mailbox, where everybody should be allowed to write but only the owner should be granted permission to read.
- $\{\mathsf{w}_\top(S), \mathsf{r}_\bot(S)\}$: the type of channels where anybody can read, but only high processes can write. This typing is typical of an information channel, where privileged users write information for everyone to read.

We give a formal definition of types in Section 5.5. Before that, we define the operational semantics and formalize a notion of security error

5.3 Reduction Semantics

The operational semantics is given, as for the π-calculus, in terms of the two relations of structural congruence and reduction. Structural congruence is defined by the following extension of the corresponding relation for the π-calculus:

Table 12 Structural congruence

π-*Calculus Rules for Structural Equivalence.*

1. $P \mid Q \equiv Q \mid P, P \mid (Q \mid R) \equiv (P \mid Q) \mid R, P \mid \mathbf{0} \equiv P$
2. $(\nu a)\mathbf{0} \equiv \mathbf{0}$, $(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$
3. $(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q$ if $a \notin fn(P)$
4. $!P \equiv !P \mid P$

Security π-*Calculus Specific Rules.*

5. $[P \mid Q]_\sigma \equiv [P]_\sigma \mid [Q]_\sigma$
6. $[(\nu x : T)P]_\sigma \equiv (\nu x : T)[P]_\sigma$
7. $[[P]_\rho]_\sigma \equiv [P]_{\sigma \sqcap \rho}$

In addition, as for the π-calculus, the definition includes the structural rules that make ≡ a congruence. The rules 5 and 6 are not surprising. In rule 7, the notation $\rho \sqcap \sigma$ indicates the *greatest lower bound* between ρ and σ in the lattice of security levels. Based on this relation, the reduction relation is defined as follows:

Table 13 Reduction Relation

(COMM)	$\overline{a}\langle \tilde{M} \rangle \mid a(\tilde{x} : \tilde{T})P \longrightarrow P\{x_1 := M_1, \ldots, x_k := M_k\}$
(COMM$_\rho$)	$[\overline{a}\langle \tilde{M} \rangle]_\sigma \mid [a(\tilde{x} : \tilde{T}).P]_\rho \longrightarrow [P\{x_1 := M_1, \ldots, x_k := M_k\}]_\rho$
(STRUCT)	$\dfrac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \quad \dfrac{P \longrightarrow P'}{[P]_\sigma \longrightarrow [P']_\sigma} \quad \dfrac{P \longrightarrow P'}{(\nu a : T)P \longrightarrow (\nu a : T)P'}$
	$\dfrac{P \equiv P' \quad P' \longrightarrow Q \quad Q' \equiv Q}{P \longrightarrow Q}$

The rule (COMM) is the asynchronous variant of the reduction rule for communications from the π-calculus. The rule (COMM$_\rho$) is the corresponding rule for processes with a clearance: as we noted, the presence of the security level does not affect the computational behavior of processes. On the other hand, it is the basis for the formalization of run-time security error.

5.4 Security as Resource Access Control

Security violations occur against a given *security policy*, which is formalized in the calculus in terms of (i) a mapping from resources (i.e. names and values) to their types, and of (ii) an auxiliary reduction relation that underlines the import of the policy by defining what it means to violate it. As a first example, given a mapping Γ, one may enforce a policy for resource access control by stating that processes at clearance σ should only have access to channels and values at security level up to (and including) σ. This can be formalized by the following additional reductions:

Table 14 Security Violation

(E-INPUT)	$[n(\tilde{x} : \tilde{T}).P]_\rho \xrightarrow{\Gamma} \mathsf{err} \quad \text{if } \mathsf{r}_\sigma(\tilde{T}) \in \Gamma(n) \Longrightarrow \sigma \not\preceq \rho$
(E-OUTPUT)	$[\overline{n}\langle M \rangle]_\rho \xrightarrow{\Gamma} \mathsf{err} \quad \text{if } \mathsf{w}_\sigma(T) \in \Gamma(n) \Longrightarrow \sigma \not\preceq \rho$
(E-OUTVAL)	$[\overline{n}\langle M \rangle]_\rho \xrightarrow{\Gamma} \mathsf{err} \quad \text{if } M : \mathsf{B}_\sigma \text{ and } \sigma \not\preceq \rho$
(E-STRUCT)	$\dfrac{P \xrightarrow{\Gamma} \mathsf{err}}{P \mid Q \xrightarrow{\Gamma} \mathsf{err}} \quad \dfrac{P \xrightarrow{\Gamma} \mathsf{err}}{[P]_\sigma \xrightarrow{\Gamma} \mathsf{err}} \quad \dfrac{P \xrightarrow{\Gamma} \mathsf{err}}{(\nu a : A)P \xrightarrow{\Gamma} \mathsf{err}}$

The rule (E-INPUT) states that a process with clearance ρ can not read from channels that are not qualified by the security policy Γ, or that have security level higher than ρ. The rule (E-OUTPUT) defines dual conditions for errors resulting from an attempt to write on restricted channels. The rule (E-OUTVAL) states that a process with clearance ρ may only communicate a value along a channel if that value is not restricted from σ-level processes. In all three cases, the security violation is signalled by a reduction to the distinguished process term err. The remaining rules are purely structural, and propagate errors from a process to its enclosing terms.

We give two examples that illustrate the import of different security policies on the reduction semantics.

Example 3 (Resource Access Violations). Consider the process

$$P \triangleq [\overline{c}\langle a\rangle]_\top \mid [c(x).\overline{x}\langle 1\rangle]_\bot.$$

consisting of a high-level and a low-level processes communicating over a channel c, for which we define the security policy Γ as follows: $\Gamma(a) = A$, $\Gamma(c) = C$. First, assume that the two types A and C are defined as follows:

$$A = \{\mathsf{w}_\top(\mathsf{int}), \mathsf{r}_\bot(\mathsf{int})\} \quad \text{and} \quad C = \{\mathsf{w}_\bot(A), \mathsf{r}_\bot(A)\}$$

By one reduction step, P reduces to the process $[\overline{a}\langle 1\rangle]_\bot$, and the latter reduces to err as a result of a low-level process attempting to write on the high-level channel a. While the violation shows up after one reduction step, it originates earlier, from the fact that the value a of "high" level type A is written to channel $c : C$ with "low" write capability. Upgrading the write capability on C does not.

Consider then defining the types A and C differently, giving C high-level write capability:

$$A = \{\mathsf{w}_\top(\mathsf{int}), \mathsf{r}_\bot(\mathsf{int})\} \quad \text{and} \quad C = \{\mathsf{w}_\top(A), \mathsf{r}_\bot(A)\}$$

Again, the reduction of P to $[\overline{a}\langle 1\rangle]_\bot$ causes a security violation (i.e. a reduction to err) because the low-level process $[\overline{a}\langle 1\rangle]_\bot$ does not have the right to write on the channel a for which the write capability is "high". Here the problem originates from the high value a being written to a channel $c : C$ with "low" read capability.

The examples give a flavor of the inherent complexity of statically enforcing a security policy. Most of this complexity is determined by "indirect" flow of information arising as a result of processes dynamically acquiring new capabilities. In our case, the intuitive and direct measures represented by the "no read-up, no write-up" slogan are not enough to guarantee the desired effects of the access control policy. Further constraints must be imposed to prevent unauthorized access: the purpose of the type system we discuss next is to provide provably sufficient conditions for absence of security violations during reduction.

5.5 Types and Subtypes

The formal definition of types is somewhat complex, as it includes well-formedness rules ensuring that types are formed according to certain consistency conditions that provide the desired security guarantees. We start defining sets of *pre-capabilities* and *pre-types*.

Pre-Capabilities

cap ::=	$\mathsf{r}_\sigma(T)$	σ-level input channel carrying values of type T
|	$\mathsf{w}_\sigma(T)$	σ-level output channel carrying values of type T

Pre-Types

S,T ::=	B_σ	base type of level σ
|	$\{cap_1,\ldots,cap_k\}$	channel type, $k \geqslant 0$
|	$(T_1,\ldots,T_k)$	tuple type, $k \geqslant 0$

Next, we introduce the consistency conditions that single out the legal set of types. The consistency conditions are formulated in terms of ordering relations over pre-capabilities and pre-types induced by the ordering on security levels. Both the subtype relations are denoted by the symbol $\leqslant$, and are the least reflexive and transitive relations that are closed under the rules below.

Table 15 Subtyping

$$\frac{T \leqslant S \quad \sigma \preceq \rho}{\mathsf{w}_\sigma(S) \leqslant \mathsf{w}_\rho(T)}\ (\textsc{Sub Output}) \qquad \frac{S \leqslant T \quad \sigma \preceq \rho}{\mathsf{r}_\sigma(S) \leqslant \mathsf{r}_\rho(T)}\ (\textsc{Sub Input})$$

$$\frac{\sigma \preceq \rho}{\mathsf{B}_\sigma \leqslant \mathsf{B}_\rho}\ (\textsc{Sub Base}) \qquad \frac{(\forall j \in J)(\exists i \in I)cap_i \leqslant cap'_j}{\{cap_i\}_{i\in I} \leqslant \{cap'_j\}_{j\in J}}\ (\textsc{Sub Type}) \qquad \frac{S_i \leqslant T_i \quad i \in [1..k]}{(S_1,\ldots,S_k) \leqslant (T_1,\ldots,T_k)}\ (\textsc{Sub Tuple})$$

The two relations are mutually recursive, following the mutually inductive definition of pre-types and pre-capabilities. The rules (Sub Input) and (Sub Output) are the direct generalization of the corresponding rules in Table 4. The remaining rules define the subtype relation over basic, channel and tuple pre-types, respectively. Note that the resulting subtype relation on pre-types generalizes the subtype relation by Pierce and Sangiorgi we discussed in Section 2.

Now the set of types (as opposed to the previously introduced pre-types) is defined by a kinding system that identifies the legal pre-types at each security level. Formally, for each level ρ, the set $Type_\rho$ is the least set closed under the following rules:

Table 16 Type Formation

$$\frac{\sigma \preceq \rho}{\mathsf{B}_\sigma \in Type_\rho}\ (\text{T-BASE}) \qquad \frac{T_i \in Type_\rho}{(T_1,\ldots,T_k) \in Type_\rho}\ (\text{T-TUPLE}) \qquad \frac{T \in Type_\sigma \quad \sigma \preceq \rho}{\{\mathsf{r}_\sigma(T)\} \in Type_\rho}\ (\text{T-RD})$$

$$\frac{T \in Type_\sigma \quad \sigma \preceq \rho}{\{\mathsf{w}_\sigma(T)\} \in Type_\rho}\ (\text{T-WR}) \qquad \frac{S \in Type_\sigma \quad T \in Type_{\sigma'} \quad \sigma,\sigma' \preceq \rho \quad S \leqslant T}{\{\mathsf{w}_\sigma(S),\mathsf{r}_{\sigma'}(T)\} \in Type_\rho}\ (\text{T-WRRD})$$

There are a number of interesting consequences of the definition that are worth pointing out. First note that if $\sigma \preceq \rho$ then $RType_\sigma \subseteq RType_\rho$. This follows by a straightforward inductive reasoning on the generation of types at each kind. The second thing to note is the compatibility requirements between the read and write capabilities in the assumptions of the rule (T-WRRD). The condition $\sigma, \sigma' \preceq \rho$ contributes to the property that $RType_\sigma \subseteq RType_\rho$ for every $\sigma \preceq \rho$. The assumption $S \leqslant T$, in turn, is a standard condition required for soundness of channel communication: any value that is written on a channel can be read from that channel at a super-type of the value's true type. Interestingly, however, the combination of this condition with the security constraints imposed by the (T-WRRD) and the other rules has also security implications.

To see them, we first state the following proposition, which can be proved by induction on the derivation of $S \in Type_\sigma$.

Proposition 3. *If $S \in Type_\sigma$, and $S \leqslant T$, then there exists ρ with $T \in Type_\rho$ and $\sigma \preceq \rho$.*

We illustrate the security implication we just mentioned with and example. Consider the type $T = \{\mathsf{w}_\top(S), \mathsf{r}_\bot(S')\}$, and a channel $a : T$. A priory, high-level processes (with clearance $\top$) may write to this channel, while low-level processes, (with clearance $\bot$) may read from it. But then, it would seem, the channel may be used to leak sensitive information, for low-level processes may read values written by high-level processes. In particular, a high-level process could write on a the name of a high-level channel: low processes could then read that name and gain access to the channel, thus resulting in a violation of the security policy induced by our instrumented semantics.

A closer look at the type formation and subtyping rules shows that this cannot happen. To see why, assume that the type T is legal, that is $T \in Type_\rho$ for some security level ρ. The hypotheses of the (T-WRRD) rule imply that $\top \preceq \rho$, hence $\rho = \top$; furthermore, the two types S and S' must be such that $S \in Type_\sigma$ with $\sigma \preceq \top$, and $S' \in Type_{\sigma'}$ with $\sigma' \preceq \bot$ and $S \leqslant S'$. From these conditions, by the above proposition, it follows that $\sigma \preceq \sigma'$, and this, together with $\sigma' \preceq \bot$, implies that $\sigma = \bot$. In other words, the formation rules require that $S \in Type_\bot$, which implies that only low values (and channels) can be written to any channel of type T. But then, even though high-level processes can write on channels of type T, they may only write low-level values: thus only low-level information may flow from high to low processes.

In their present form, the type formation rules limit types to contain at most one read and one write capabilities: this clearly results in a loss of expressive power, but there is no fundamental difficulty in extending the formalization to handle types in the general form.

5.6 Typing Rules

The typing rules, given in Table 17, derive judgments in two forms: the usual form $\Gamma \vdash M : T$ stating that term M has type T, and the form $\Gamma \vdash^\sigma P$ which says that process P is well-typed in the context Γ, at security level σ (the rules for parallel composition and replication are standard, and omitted).

Table 17 Typing Rules for the Security π-calculus

$$\frac{\Gamma(u) \leqslant A}{\Gamma \vdash u : A}\ (\textsc{Name}) \qquad \frac{\Gamma, a : T \vdash^{\sigma} P \quad T \in Type_{\sigma} \quad a \notin \mathsf{Dom}(\Gamma)}{\Gamma \vdash^{\sigma} (\nu a : T)P}\ (\textsc{Restr}) \qquad \frac{\Gamma \vdash^{\sigma \sqcap \rho} P}{\Gamma \vdash^{\sigma} [P]_{\rho}}\ (\textsc{Proc})$$

$$\frac{\Gamma, \tilde{x} : \tilde{T} \vdash^{\sigma} P \quad \Gamma \vdash u : \mathsf{r}_{\sigma}(\tilde{T}) \quad \tilde{x} \cap \mathsf{Dom}(\Gamma) = \varnothing}{\Gamma \vdash^{\sigma} u(\tilde{x} : \tilde{T}).P}\ (\textsc{Input}) \qquad \frac{\Gamma \vdash u : \mathsf{w}_{\sigma}(T) \quad \Gamma \vdash M : T}{\Gamma \vdash^{\sigma} \overline{u}\langle M \rangle}\ (\textsc{Output})$$

The first three rules should be self-explanatory, but note, in the (RESTR) rule, that only names at level (at most) σ may legally be introduced by well-typed processes running at clearance σ. In the (INPUT) rule, the premises guarantee that the channel is used consistently with its associated capabilities and security level. For the latter, note that u offers a read capability at the same level σ at which the input process is currently running. From the definition of subtyping, and the rule (NAME), it follows that $\Gamma(u) = T$ for a type T that includes a read capability at level $\rho \preceq \sigma$: this guarantees that a process with clearance σ may read from any channel with security level up-to σ, as desired. The same reasoning applies to the (OUPUT) rule. The constraints imposed by the typing rules, together with the constraints imposed on the type formation rules provide static guarantees of type safety, that is absence of run-time violations for every well-typed process. Type safety is formalized as follows.

Theorem 8 (Type Safety for Resource Access). $\Gamma \vdash^{\sigma} P$ *implies* $[P]_{\sigma} \xrightarrow{\Gamma} \not\rightarrow \mathsf{err}$

In other words, if a process P is well-typed at clearance σ, then neither P nor any of its derivatives will attempt a non-authorized access to a value or a channel restricted from level σ. That P is free of error reductions follows directly from the above theorem: that it is also true of the derivatives of P follows by the fact that well-typedness at any clearance level is preserved by reduction as stated by the following theorem.

Theorem 9 (Subject Reduction). *If* $\Gamma \vdash^{\sigma} P$ *and* $P \longrightarrow Q$, *then* $\Gamma \vdash^{\sigma} Q$

To exemplify the impact of the type system in enforcing our policy of resource access control, consider the process

$$P \triangleq (\nu a : A)(\nu c : C)[\overline{c}\langle a \rangle]_{\top} \mid [c(x).\overline{x}\langle 1 \rangle]_{\bot}.$$

In Example 3 we discussed two definitions for the types A and C: in both cases, the process is ill-typed independently of the clearance ($\top$ or $\bot$) at which we may type it. In fact, ill-typedness is a consequence of the type C being ill-formed, as $A \in Type_{\top}$ may not be read from channels of type C with $\bot$-level read capability.

We give more examples illustrating the rôle of types for security in the next section, where we discuss a variation of the type system that provides guarantees for the "no read-up, no write-down" constraints distinctive of the *defense* policy of MAC security.

5.7 MAC Policies: *Defense* Security

Few changes are required to type formation rules to account for this case of MAC security: the typing rules, instead, are unchanged. To understand and motivate the changes, we start with a simple examples.

Example 4 (Defense Security). Consider again the process P from example 3:

$$P \triangleq [\overline{c}\langle l\rangle]_{\top} \mid [c(x).\overline{x}\langle 1\rangle]_{\bot}.$$

where $\Gamma(c) = C$ and $\Gamma(l) = L$, and the two types C and L are defined as follows.

$$C = \{\mathsf{w}_{\bot}(L), \mathsf{r}_{\bot}(L)\} \quad \text{and} \quad L = \{\mathsf{w}_{\bot}(\mathsf{int}), \mathsf{r}_{\bot}(\mathsf{int})\}.$$

With the current type system, P is well typed in Γ, as the channel c has "low" type and offers read and write capabilities: hence both processes may legally access c. The same is true of the type assignment $l : L$, and $c : C$ with L as above, and C defined now as $C = \{\mathsf{w}_{\top}(L), \mathsf{r}_{\bot}(L)\}$. Indeed, it is not difficult to see that there is no violation of our resource access policy, as there is no P error reduction for P or any of its derivatives.

In both the above cases, the term P would be rejected as "unsafe" under defense security, as in both cases a high-level process ends-up writing a low-level object, hence establishing a high-to-low flow of information. It is, however, easy to identify the source of the problems, and change the type system to enforce the new constraints.

In the first case, the problem is a direct violation of the "no write-down" constraint, which results from the current definition of subtyping. The judgment $\Gamma \vdash^{\top} \overline{c}\langle l\rangle$ is derivable by an application of the (OUTPUT) from the premise $\Gamma \vdash c : \mathsf{w}_{\top}(L)$, as $\Gamma(c) \leqslant \mathsf{w}_{\top}(L)$. In particular, the subtype relation holds because so does $\mathsf{w}_{\bot}(L) \leqslant \mathsf{w}_{\top}(L)$: to prevent the write-down, it is thus enough to rule out the latter relation.

In the second case, instead, the problem results from the channel c offering a write capability to processes running at high clearance, and read capability to low processes. As a result, a low process can "read up" information written by high-level processes on the same channel. To prevent such situations, it is enough to refine the type formation rules by requiring that a read capability on a channel type not be lower than the write capability (if any).

The new set of types may thus be defined as follows:

Definition 2 (Types for defense security). *For any security level* ρ*, let* $Type_{\rho}$ *be the least set of types that is closed under the subtype and kind rules of Section 5.5, where*

- *rule (*SUB OUTPUT*) is replaced by:* $\dfrac{T \leqslant S}{\mathsf{w}_{\sigma}(S) \leqslant \mathsf{w}_{\sigma}(T)}$
- *rule (*T-WRRD*) is replaced by:* $\dfrac{S \in Type_{\sigma} \; T \in Type_{\sigma'} \quad \sigma \preceq \sigma' \preceq \rho \quad S \leqslant T}{\{\mathsf{w}_{\sigma}(S), \mathsf{r}_{\sigma'}(T)\} \in Type_{\rho}}$

Given the new definition of types, and the typing rules of Table 17, it is possible to show that well-typed processes do not cause any defense security violation. Of course, this requires a new definition of error reductions, to reflect the desired notion of violation under defense security.

5.8 Information Flow Security

Having outlined a solution to defense security, we conclude our discussion on the security π-calculus with a few observations on information-flow security.

As we already mentioned, information flow security aims at protecting confidentiality and integrity of information by preventing 'implicit' flow of information via covert channels. Examples of covert channels may naturally formalized in the security π-calculus.

As a first example, consider the system:

$$[h(x).\mathsf{if}\ x = 0\ \mathsf{then}\ \overline{hl}\langle 0\rangle\ \mathsf{else}\ \overline{hl}\langle 1\rangle]_\top \mid [hl(z).Q]_\bot$$

where one has $hl : HL$ and $h : H$, and the two types in question are defined as follows:

$$HL = \{\mathsf{w}_\top(\mathsf{int}), \mathsf{r}_\bot(\mathsf{int})\}, \quad H = \{\mathsf{w}_\top(\mathsf{int}), \mathsf{r}_\top(\mathsf{int})\}.$$

We have already noticed the presence of information flow in a similar process in Example 4, resulting from a low process reading on a channel that is written by a high process. Here the case of information flow is more interesting, however, as the low process gains additional information on the value x transmitted over the high-channel h. Indeed, the example is not problematic, as the definition of types for defense security rules out this system as insecure. Consider however, the new system:

$$[h(x)\,\mathsf{if}\ x = 0\ \mathsf{then}\ [\overline{l}\langle 0\rangle]_\bot\ \mathsf{else}\ [\overline{l}\langle 1\rangle]_\bot]_\top \mid [l(z).Q]_\bot$$

where now $h : H$, $l : L$ and the two types are defined as follows:

$$H = \{\mathsf{w}_\top(\mathsf{int}), \mathsf{r}_\top(\mathsf{int})\}, \quad L = \{\mathsf{w}_\bot(\mathsf{int}), \mathsf{r}_\bot(\mathsf{int})\}$$

This system is well-typed, even with the type system of Section 5.7, as the high-level process downgrades itself prior to writing on the low-level channel l. Still, the system exhibits the same high-to-low flow of information as before.

As a final example, it is instructive to look at the impact of synchronous communication over information flow. Assuming synchronous communication the following has the same problems as the previous one. Consider

$$[\overline{l_1}\langle\rangle.Q_1 \mid \overline{l_2}\langle\rangle.Q_2]_\bot \quad \mid \quad [\mathsf{if}\ x = 0\ \mathsf{then}\ l_1()\ \mathsf{else}\ l_2()]_\top$$

Assuming $L = \{\mathsf{w}_\bot(), \mathsf{r}_\bot()\}$, and $l_1, l_2 : L$, the system is well-typed, and yet there is an implicit flow of information arising purely from synchronization: information on the value of x may be assumed by both the continuations Q_1 and Q_2 of the low process.

5.9 Further Reading

The work on information-flow security for the π-calculus is well developed in [34] and subsequent work by Hennessy[2]. A related approach is discussed by Honda and Vasconcelos in [35].

[2] (see `http://www.cogs.susx.ac.uk/users/matthewh/research.html`).

Information flow analysis based on non-interference originated with the seminal idea of Goguen and Meseguer [32]. In process calculi, the first formalizations of non-interference were proposed in [51, 24, 52], based on suitable trace-based notions of behavioral process equivalence for CCS-like calculi.

Information flow analyses based on typing techniques were first discussed in the pioneering work D. and P. Dennings [19], in which a type system detecting direct and indirect flows among program variables in imperative languages was devised. This initial idea was refined and formalized some twenty years later in work on type systems providing guarantees of non-interference in multi-threaded programming languages both in nondeterministic [56, 55] and probabilistic settings [53].

Type systems for secure information flow and non-interference in process have also been applied to enforce secrecy of cryptographic protocols. The most notable applications of typing techniques for analysis of security protocols have been developed for Abadi and Gordon's *spi* calculus [5, 10], that we discuss in the next section.

6 The CryptoSPA Calculus

In this section we report from [27] the *Cryptographic Security Process Algebra* (CryptoSPA for short). It is basically a variant of value-passing CCS [41], where the processes are provided with some primitives for manipulating messages. In particular, processes can perform message encryption and decryption, and also construct complex messages by composing together simpler ones.

6.1 Syntax of the Calculus

CryptoSPA syntax is based on the following elements:

- A set $I = \{a, b, \ldots\}$ of *input* channels, a set $O = \{\bar{a}, \bar{b}, \ldots\}$ of *output* ones;
- A set M of basic messages and a set K of encryption keys with a function $\cdot^{-1} : K \rightarrow K$ such that $(k^{-1})^{-1} = k$. The set $\mathcal{M}$ of all messages is defined as the least set such that $M \cup K \in \mathcal{M}$ and $\forall m \in \mathcal{M}$, $\forall k \in K$ we have that (m, m') and $\{m\}_k$ also belong to $\mathcal{M}$;
- A set C of *public* channels; these channels represent the insecure network where the enemy can intercept and fake messages;
- A family $\mathcal{U}$ of sets of messages and a function $Msg(c) : I \cup O \longrightarrow \mathcal{U}$ which maps every channel c into the set of possible messages that can be sent and received along such a channel. Msg is such that $Msg(c) = Msg(\bar{c})$.
- A set $Act = \{c(m) \mid c \in I, m \in Msg(c)\} \cup \{\overline{c}\langle m\rangle \mid c \in O, m \in Msg(c)\} \cup \{\tau\}$ of actions (τ is the internal, invisible action), ranged over by a; we also have a function $chan(a)$ which returns c if a is either $c(m)$ or $\overline{c}\langle m\rangle$, and the special channel *void* when $a = \tau$; we assume that *void* is never used within a restriction operator (see below).
- A set *Const* of constants, ranged over by A.

The syntax of CryptoSPA agents is defined as follows:

$$E ::= \mathbf{0} \mid c(x).E \mid \overline{c}\langle e\rangle.E \mid \tau.E \mid E + E \mid E \,\|\, E \mid E \setminus L \mid E[f] \mid$$
$$\mid A(m_1, \ldots, m_n) \mid [e = e']E;E \mid [\langle e_1 \ldots e_r\rangle \vdash_{rule} x]E;E$$

where x is a variable, $m_1,\ldots,m_n$ are messages, $e,e_1,\ldots,e_r$ are messages (possibly containing variables) and L is a set of input channels. Both the operators $c(x).E$ and $[\langle e_1 \ldots e_r\rangle \vdash_{rule} x]E;E'$ bind the variable x in E. It is also necessary to define constants as follows: $A(x_1,\ldots,x_n) \triangleq E$ where E is a CryptoSPA agent which may contain no free variables except $x_1,\ldots,x_n$, which must be distinct.

Besides the standard value-passing CCS operators, we have an additional one that has been introduced in order to model message handling and cryptography. Informally, the $[\langle m_1 \ldots m_r\rangle \vdash_{rule} x]E_1;E_2$ process tries to deduce an information z from the tuple of messages $\langle m_1 \ldots m_r\rangle$ through one application of rule $\vdash_{rule}$; if it succeeds then it behaves like $E_1[z/x]$, otherwise it behaves like E_2; for example, given a rule $\vdash_{dec}$ for decryption, process $[\langle\{m\}_k,k^{-1}\rangle \vdash_{dec} x]E_1;E_2$ decrypts message $\{m\}_k$ through key k^{-1} and behaves like $E_1[m/x]$ while $[\langle\{m\}_k,k'\rangle \vdash_{dec} x]E_1;E_2$ (with $k' \neq k^{-1}$) tries to decrypt the same message with the wrong inverse key k' and (since it is not permitted by $\vdash_{dec}$) it behaves like E_2.

We call $\mathcal{E}$ the set of all the CryptoSPA terms, and we define $sort(E)$ to be the set of all the channels syntactically occurring in the term E.

6.2 The Operational Semantics of CryptoSPA

In order to model message handling and cryptography, in Table 18 we define an inference system which formalizes the way messages may be manipulated by processes.

Table 18 Inference System for message manipulation

Let $m,m' \in \mathcal{M}$ and $k,k^{-1} \in K$.

$$\frac{\text{m \& m'}}{\text{(m,m')}}\ (\vdash_{pair}) \qquad \frac{\text{(m,m')}}{\text{m}}\ (\vdash_{fst}) \qquad \frac{\text{(m,m')}}{\text{m'}}\ (\vdash_{snd})$$

$$\frac{\text{m \& k}}{\{m\}_k}\ (\vdash_{enc}) \qquad \frac{\{m\}_k \ \&\ k^{-1}}{\text{m}}\ (\vdash_{dec})$$

It is indeed quite similar to those used by many authors (see, e.g., [38, 39]). In particular it can combine two messages obtaining a pair (rule $\vdash_{pair}$); it can extract one message from a pair (rules $\vdash_{fst}$ and $\vdash_{snd}$); it can encrypt a message m with a key k obtaining $\{m\}_k$ and finally decrypt a message of the form $\{m\}_k$ only if it has the corresponding (inverse) key k^{-1} (rules $\vdash_{enc}$ and $\vdash_{dec}$). We denote with $\mathcal{D}(\phi)$ the set of messages that can be deduced by applying the inference rules on the messages in ϕ. Note that we are assuming encryption as completely reliable. Indeed we do not allow any kind of cryptographic attack, e.g., the guessing of secret keys. This permits to observe the attacks that can be carried out even if cryptography is completely reliable.

The formal behavior of a CryptoSPA term is described by means of the *labelled transition system* $< \mathcal{E}, Act, \{\xrightarrow{a}\}_{a\in A} >$, where $\xrightarrow{a}_{a\in A}$ is the least relation between CryptoSPA terms induced by axioms and inference rules of Table 19.

Table 19 Operational semantics

$$(input)\frac{m \in Msg(c)}{c(x).E \xrightarrow{c(m)} E[m/x]} \quad (output)\frac{m \in Msg(c)}{\overline{c}\langle m\rangle.E \xrightarrow{\overline{c}\langle m\rangle} E} \quad (internal)\frac{}{\tau.E \xrightarrow{\tau} E}$$

$$(\|_1)\frac{E \xrightarrow{a} E'}{E \,\|\, E_1 \xrightarrow{a} E' \,\|\, E_1} \qquad (\|_2)\frac{E \xrightarrow{c(m)} E' \quad E_1 \xrightarrow{\overline{c}\langle m\rangle} E_1'}{E \,\|\, E_1 \xrightarrow{\tau} E' \,\|\, E_1'}$$

$$(+_1)\frac{E \xrightarrow{a} E'}{E + E_1 \xrightarrow{a} E'} \quad ([f])\frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]} \quad (\backslash L)\frac{E \xrightarrow{a} E' \;\; chan(a) \notin L}{E\backslash L \xrightarrow{a} E'\backslash L}$$

$$(=_1)\frac{m \neq m' \quad E_2 \xrightarrow{a} E_2'}{[m = m']E_1; E_2 \xrightarrow{a} E_2'} \qquad (=_2)\frac{m = m' \quad E_1 \xrightarrow{a} E_1'}{[m = m']E_1; E_2 \xrightarrow{a} E_1'}$$

$$(def)\frac{E[m_1/x_1, \ldots, m_n/x_n] \xrightarrow{a} E' \quad A(x_1, \ldots, x_n) \triangleq E}{A(m_1, \ldots, m_n) \xrightarrow{a} E'}$$

$$(\mathcal{D}_1)\frac{\langle m_1 \ldots m_r\rangle \vdash_{rule} m \quad E_1[m/x] \xrightarrow{a} E_1'}{[\langle m_1 \ldots m_r\rangle \vdash_{rule} x]E_1; E_2 \xrightarrow{a} E_1'} \quad (\mathcal{D}_2)\frac{\nexists m : \langle m_1 \ldots m_r\rangle \vdash_{rule} m \quad E_2 \xrightarrow{a} E_2'}{[\langle m_1 \ldots m_r\rangle \vdash_{rule} x]E_1; E_2 \xrightarrow{a} E_2'}$$

Plus symmetric rules for $+_1$, $\|_1$ and $\|_2$are omitted

Example. We present a very simple example of a protocol where A sends a message m_A to B encrypted with a key k_{AB} shared between A and B[3]. We define it as $P \triangleq A(m_A, k_{AB}) \,\|\, B(k_{AB})$ where $A(m,k) \triangleq \overline{c}\langle\{m\}_k\rangle$[4] and $B(k) \triangleq c(y).[\langle y,k\rangle \vdash_{dec} z]\overline{out}\langle z\rangle$. Moreover, $k_{AB}^{-1} = k_{AB}$ (symmetric encryption) and $Msg(c) = \{\{m\}_k \mid m \in M, k \in K\}$. We want to analyze the execution of P with no intrusions, we thus consider $P \setminus \{c\}$, since the restriction guarantees that c can be used only inside P. We obtain a system which can only execute action $\overline{out}\langle m_A\rangle$ that represents the correct transmission of m_A from A to B. In particular, the only possible execution is the one where A sends to B message $\{m_A\}_{k_{AB}}$ and then $\overline{out}\langle m_A\rangle$ is executed:

$$P \setminus \{c\} \xrightarrow{\tau} (\mathbf{0} \parallel [\langle\{m_A\}_{k_{AB}}, k_{AB}\rangle \vdash_{dec} z]\overline{out}\langle z\rangle) \setminus \{c\} \xrightarrow{\overline{out}\langle m_A\rangle} (\mathbf{0} \,\|\, \mathbf{0}) \setminus \{c\}$$

The calculus of CryptoSPA has been successfully applied to the automatic specification and the verification of security protocols, see [20, 26, 27, 21, 25, 23, 48–50, 22].

[3] For the sake of readability, we omit the termination $\mathbf{0}$ at the end of every agent specifications, e.g., we write a in place of $a.\mathbf{0}$. We also write $[m = m']E$ in place of $[m = m']E;\mathbf{0}$ and analogously for $[\langle m_1 \ldots m_r\rangle \vdash_{rule} x]E;\mathbf{0}$.

[4] Note that this process could be also written as $A(m,k) \triangleq [\langle m,k\rangle \vdash_{enc} x]\overline{c}\langle x\rangle$.

7 The Spi-Calculus

The spi calculus is an extension of the pi calculus with cryptographic primitives that has been introduced by Abadi and Gordon in [5, 10], The spi calculus is designed for describing and analyzing security protocols, such as those for authentication and for electronic commerce. These protocols rely on cryptography and on communication channels with properties like authenticity and privacy. Accordingly, cryptographic operations and communication through channels, are the main ingredients of the spi calculus.

As we discussed in Section 1, some abstract security protocol can be expressed in the pi calculus, thanks to its simple but powerful primitives for channels. Moreover, the scoping rules of the pi calculus guarantee that the environment of a protocol (the attacker) cannot access a channel that is not explicitly given; scoping is thus the basis of security. However, as we pointed out, when considering a distributed environment, it is not realistic to rely only on the scope rules, we also have to prevent the context from having free access to public channels over which private names are communicated. In a distributed environment such a channel protection relies on the use of cryptography. With shared-key cryptography, secrecy can be achieved by communication on public channels under secret keys.

The spi calculus is thus an extension of the pi calculus that consider cryptographic issues in more detail. Its features can be summarized as follows:

- it permits an explicit representation of the use of cryptography in protocols, while it does not seem easy to represent encryption and decryption within the pi calculus;
- it relies on the powerful scoping constructs of the pi calculus;
- within the spi calculus, the environment can be defined as an arbitrary spi calculus process instead of giving an explicit model;
- security properties, both integrity and secrecy, can be represented as equivalences and analyzed by means of static techniques.

7.1 Syntax and Semantics

The syntax of the spi calculus extends a particular version of the pi calculus with constructs for encrypting and decrypting messages (see Table 20) . In standard pi calculus names are the only terms. For convenience, the syntax of spi calculus also contains constructs for paring and numbers, namely (M,N), 0 and $succ(M)$. Furthermore, the term $\{M_1,\ldots,M_k\}_N$ represents the ciphertext obtained by encrypting $M_1,\ldots,M_k$ under the key N using a shared-key cryptosystem such as DES. The key is an arbitrary term; typically, names are used as keys because in the spi calculus names are unguessable capabilities.

Intuitively, the new constructs of spi calculus have the following meanings: a match $[M\ is\ N]P$ behaves as P provided that terms M and N are the same, otherwise it is stuck. A pair splitting process $let\ (x,y) = M\ in\ P$, where x and y are bound in P, behaves as $P\{x := N, y := L\}$ if the term M is the pair (N,L). An integer case process $case\ M\ of\ 0 : P\ succ(x) : Q$, where x is bound in Q, behaves as P if term M is 0, as $Q\{x := N\}$ if M is $succ(N)$. Finally the process $case\ L\ of\ \{x_1,\ldots,x_k\}_N\ in\ P$, where $x_1,\ldots,x_k$ are bound

Table 20 Spi calculus syntax

Expressions	L,M,N	$::= bv$	basic value
		$\mid a,\ldots,p$	name
		$\mid x,\ldots,z$	variable
		$\mid (M,N)$	pair
		$\mid 0$	zero
		$\mid succ(M)$	successor
		$\mid \{M_1,\ldots,M_k\}_N$	shared-key encryption ($k \geq 0$)
Processes	P,Q,R	$::= \mathbf{0}$	stop
		$\mid \overline{u}\langle N_1,\ldots,N_k\rangle.P$	output ($k \geq 0$)
		$\mid u(x_1,\ldots,x_k).P$	input ($k \geq 0$)
		$\mid (\nu a)P$	restriction
		$\mid P \mid P$	composition
		$\mid !P$	replication
		$\mid [M \textit{ is } N]P$	match
		$\mid \textit{let } (x,y) = M \textit{ in } P$	pair splitting
		$\mid \textit{case } M \textit{ of } 0 : P \; succ(x) : Q$	integer case
		$\mid \textit{case } L \textit{ of } \{x_1,\ldots,x_k\}_N \textit{ in } P$	shared-key decryption ($k \geq 0$)

in P, attempts to decrypt the term L with the key N; if L is a ciphertext of the form $\{M_1,\ldots,M_k\}_N$, then the process behaves as $P\{x_1 := M_1,\ldots,x_k := M_k\}$, and otherwise the process is stuck.

Implicit in the definition of the spi calculus syntax are some standard but significant assumptions about cryptography: (*i*) the only way to decrypt an encrypted packet is to know the corresponding key; (*ii*) an encrypted packet does not reveal the key that was used to encrypt it; (*iii*) there is sufficient redundancy in messages so that the decryption algorithm can detect whether a ciphertext was encrypted with the expected key.

Operational Semantics. The operational semantics of spi calculus can be defined in terms of a structural congruence and a reduction relation, extending the corresponding relations defined in Section 1 for the π calculus. In particular, structural congruence is defined as the least congruence relation closed under rules 1.-4. of Section 1.1 plus the following rules:

(Red Repl)	$!P$	$\equiv P \mid !P$
(Red Match)	$[M \textit{ is } M]P$	$\equiv P$
(Red Let)	$\textit{let } (x,y) = (M,N) \textit{ in } P$	$\equiv P\{x := M, y := N\}$
(Red Zero)	$\textit{case } 0 \textit{ of } 0 : P \; succ(x) : Q$	$\equiv P$
(Red Succ)	$\textit{case } succ(M) \textit{ of } 0 : P \; succ(x) : Q$	$\equiv Q\{x := M\}$
(Red Decrypt)	$\textit{case } \{\tilde{M}\}_N \textit{ of } \{\tilde{x}\}_N \textit{ in } P$	$\equiv P\{\tilde{x} := \tilde{M}\}$

The reduction relation is then the least relation closed under the following rules: In order to develop proof techniques for the spi calculus, we define an auxiliary, equivalent, operational semantics based on a commitment relation, in the style of Milner [44]. The

Table 21 Reduction Relation

$$\text{(COMM)}\quad n(x_1,\ldots,x_k).P \mid \overline{n}\langle M_1,\ldots,M_k\rangle.Q \longrightarrow P\{x_1 := M_1,\ldots,x_k := M_k\} \mid Q$$

$$\text{(STRUCT)}\quad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \frac{P \longrightarrow P'}{(\nu n)P \longrightarrow (\nu n)P'} \qquad \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$$

definition of commitment depends on two new syntactic forms: *abstractions* and *concretions*. An abstraction is a term of the form $(\tilde{x})P$, where $x_1,\ldots,x_k$ are bound variables, and P is a process. A concretion is a term of the form $(\nu\tilde{m})\langle\tilde{M}\rangle P$ where $M_1,\ldots,M_k$ are expressions, P is a process, and the names $m_1,\ldots m_l$ are bound in $M_1,\ldots,M_k$ and P. Finally an *agent* is an abstraction, a process or a concretion. We use the metavariables A and B to stand for arbitrary agents, C for concretions, and F for abstractions.
Restriction and parallel composition for abstractions and concretions are defined as follows:

$$\begin{array}{lll}
(\nu m)(\tilde{x})P & = (\tilde{x})(\nu m)P & \\
Q \mid (\tilde{x})P & = (\tilde{x})(Q \mid P) & \text{with } \{\tilde{x}\} \cap fv(Q) = \emptyset \\
(\nu m)(\nu\tilde{n})\langle\tilde{M}\rangle P & = (\nu m,\tilde{n})\langle\tilde{M}\rangle P & \text{with } m \notin \{\tilde{n}\} \\
Q \mid (\nu\tilde{n})\langle\tilde{M}\rangle P & = (\nu\tilde{n})\langle\tilde{M}\rangle Q \mid P & \text{with } \{\tilde{n}\} \cap fn(Q) = \emptyset
\end{array}$$

If F is the abstraction $(x_1,\ldots,x_k)P$ and C is the concretion $(\nu n_1,\ldots,n_l)\langle M_1,\ldots,M_k\rangle Q$, and if $\{n_1,\ldots,n_l\} \cap fn(P) = \emptyset$, we define the process $F@C$ and $C@F$ as follows:

$$\begin{array}{l}
F@C \triangleq (\nu n_1)\ldots(\nu n_l)(P\{x_1 := M_1,\ldots,x_k := M_k\} \mid Q) \\
C@F \triangleq (\nu n_1)\ldots(\nu n_l)(Q \mid P\{x_1 := M_1,\ldots,x_k := M_k\})
\end{array}$$

Let the *reduction relation* $>$ be the least relation on closed processes that satisfies the following axioms:

$$\begin{array}{lll}
\text{(Red Repl)} & !P & > P \mid !P \\
\text{(Red Match)} & [M \textit{ is } M]P & > P \\
\text{(Red Let)} & \textit{let } (x,y) = (M,N) \textit{ in } P & > P\{x := M, y := N\} \\
\text{(Red Zero)} & \textit{case } 0 \textit{ of } 0 : P \textit{ succ}(x) : Q & > P \\
\text{(Red Succ)} & \textit{case succ}(M) \textit{ of } 0 : P \textit{ succ}(x) : Q & > Q\{x := M\} \\
\text{(Red Decrypt)} & \textit{case } \{\tilde{M}\}_N \textit{ of } \{\tilde{x}\}_N \textit{ in } P & > P\{\tilde{x} := \tilde{M}\}
\end{array}$$

A *barb* β is a name m (representing input) or a co-name $\overline{m}$ (representing output). An *action* is a barb or a distinguished *silent action* τ. The commitment relation is written $P \xrightarrow{\alpha} A$ where P is a closed process, α is an action and A is a closed agent. The commitment relation is defined by rules in Table 22.

The following proposition asserts that the two operational semantics for spi calculus, the one based on reduction relation, and the other one based on commitment relation, are equivalent.

Proposition 4. *$P \longrightarrow Q$ if and only if $P \xrightarrow{\tau} \equiv Q$.*

Table 22 Commitment Relation

$$\text{(COMM OUT)}\quad \frac{}{\overline{m}\langle\tilde{M}\rangle.P \xrightarrow{\overline{m}} (\nu)\langle\tilde{M}\rangle P} \qquad \text{(COMM IN)}\quad \frac{}{m(\tilde{x}).P \xrightarrow{m} (\tilde{x})P}$$

$$\text{(COMM INTER 1)}\quad \frac{P \xrightarrow{m} F \quad Q \xrightarrow{\overline{m}} C}{P \mid Q \xrightarrow{\tau} F@C}$$

$$\text{(COMM INTER 2)}\quad \frac{P \xrightarrow{\overline{m}} C \quad Q \xrightarrow{m} F}{P \mid Q \xrightarrow{\tau} C@F} \qquad \text{(COMM PAR 1)}\quad \frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q} \qquad \text{(COMM PAR 2)}\quad \frac{Q \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} P \mid A}$$

$$\text{(COMM RES)}\quad \frac{P \xrightarrow{\alpha} A \quad \alpha \notin \{m,\overline{m}\}}{(\nu m)P \xrightarrow{\alpha} (\nu m)A} \qquad \text{(COMM RED)}\quad \frac{P > Q \quad Q \xrightarrow{\alpha} A}{P \xrightarrow{\alpha} A}$$

Testing Equivalence. Testing equivalence is useful to compare process behaviors and to define security properties such as secrecy and authentication.

Let a *test* be a pair (Q,β) consisting of a closed process Q and a barb β. We say that P *passes* a test (Q,β) if and only if

$$(P \mid Q) \xrightarrow{\tau} Q_1 \ldots \xrightarrow{\tau} Q_n \xrightarrow{\beta} A$$

for some $n \geq 0$, some processes $Q_1,\ldots,Q_n$ and some agent A. We obtain a testing preorder $\sqsubseteq$ and a testing equivalence $\simeq$ on closed processes:

$$P \sqsubseteq P' \triangleq \text{for any test } (Q,\beta), \text{ if } P \text{ passes } (Q,\beta) \text{ then } P' \text{ passes } (Q,\beta)$$
$$P \simeq P' \triangleq P \sqsubseteq P' \text{ and } P' \sqsubseteq P$$

The idea of testing equivalence comes from the work of De Nicola and Hennessy [18]. A test neatly formalizes the idea of a generic experiment or observation that another process (such as an attacker) might perform on a process. Thus testing equivalence concisely captures the concept of equivalence in an arbitrary environment. Furthermore, testing equivalence is a congruence; more precisely, if $P \simeq Q$ then P and Q may be used interchangeably in any context, that is $\mathcal{C}[P] \simeq \mathcal{C}[Q]$ for any closed context $\mathcal{C}$.

7.2 Secrecy by Typing in the Spi Calculus

In this section we describe rules that Abadi proposed in [1] for achieving secrecy properties in security protocols expressed in the spi calculus. The rules have the form of typing rules; they guarantee that, if a protocol typechecks, then it does not leak its secret inputs. Before starting the formalization of the type system, we recall from [1] some informal security principle we adopt in the following.

First, our rules should constrain only the principals that want to protect their secrets form the attacker. That is since in some situations we may assume that the attacker cannot guess certain keys, but we cannot expect to restrict the code that the attacker runs.

We then consider only three classes of data: *Public* data, which can be communicated to anyone, *Secret* data, which should not be leaked, *Any* data, that is, arbitrary data. We refer to *Secret, Public* and *Any* as levels or types. We then assume that

> The result of encrypting data with a public key has the same classification as the data, while the result of encrypting data with a secret key may be made public.
>
> Only public data can be sent on public channels, while all kinds of data may be sent on secret channels.

Because a piece of data of level *Any* could be of level *Secret*, it should not be leaked. On the other hand, a piece of data of level *Any* could be of level *Public*, so it cannot be used as a secret. Thus

> if all we know about a piece of data is that it has level *Any*, then we should protect it as if it had level *Secret*, but we can exploit it only if it had level *Public.*

In our rules we adopt a standard format for all messages on secret channels or under secret keys. Each message on a secret channel has three components, the first of which has level *Secret*, the second *Any*, and the third *Public*, plus a confounder component. This schema implements the following principle:

> Upon receipt of a message, it should be easy to decide which part of the contents are sensitive information, if any. This decision is least error-prone when it does not depend on implicit context.

For the use of confounders, note that if each encrypted message of a protocol includes a freshly generated confounder in a standard position, then the protocol will not generate the same ciphertext more than once.

Types and Typing Rules. The syntax of types corresponds to the three classes of data:

$$\textit{Types} \quad S,T \quad ::= \quad \textit{Public} \mid \textit{Secret} \mid \textit{Any}$$

There is also a subtyping relation between types: $T <: S$ holds if T equals S or if S is *Any*. The typing system contains three forms of judgments: $\vdash E$ stating that the environment E is well formed, $E \vdash M : T$ stating that the term M is of level T in E, and $E \vdash P$ stating that the process P typechecks in E.

An environment is a list of distinct names and variables with associated levels. In addition, each name n has an associated term of the form $\{M_1,\ldots,M_k,n\}_N$. This association means that the name n may be used as a confounder only in the term $\{M_1,\ldots,M_k,n\}_N$. We write $x : T$ for variable x with level T, and $n : T :: \{M_1,\ldots,M_k,n\}_N$. The rules for environments are in Table 23.

The hypotheses of rule (ENV NAME) imply that if a variable x occurs in $\{M_1,\ldots,M_k,n\}_N$, then it is declared in E. This means that we cannot instantiate the variable x in several ways, obtaining several different terms with the same confounder, and thus defeating the purpose of confounders.

Table 23 Environment Formation

$$\text{(ENV } \emptyset\text{)}\ \frac{}{\vdash \emptyset} \qquad \text{(ENV VAR)}\ \frac{\vdash E \quad x \notin dom(E)}{\vdash E, x : T} \qquad \text{(ENV NAME)}\ \frac{\vdash E \quad n \notin dom(E) \quad E \vdash M_i : T_i \ \ i = 1..k \quad E \vdash N : S}{\vdash E, n : T :: \{M_1, \ldots, M_k, n\}_N}$$

Table 24 Typing Rules for Terms

$$\text{(SUBSUM)}\ \frac{E \vdash M : T \quad T <: S}{E \vdash M : S} \qquad \text{(VARIABLE)}\ \frac{\vdash E \quad x : T \in E}{E \vdash x : T} \qquad \text{(NAME)}\ \frac{\vdash E \quad n : T :: \{M_1, \ldots, M_k, n\}_N \text{ in } E}{E \vdash n : T}$$

$$\text{(ZERO)}\ \frac{\vdash E}{E \vdash 0 : \mathit{Public}} \qquad \text{(SUCC)}\ \frac{E \vdash M : T}{E \vdash \mathit{succ}(M) : T} \qquad \text{(PAIR)}\ \frac{E \vdash M : T \quad E \vdash N : T}{E \vdash (M, N) : T}$$

$$\text{(ENCRYPT } \mathit{Secret}\text{)}\ \frac{E \vdash M_1 : \mathit{Secret} \ \ E \vdash M_2 : \mathit{Any} \quad E \vdash M_3 : \mathit{Public} \quad E \vdash N : \mathit{Secret} \quad n : T :: \{M_1, M_2, M_3, n\}_N \text{ in } E}{E \vdash \{M_1, M_2, M_3, n\}_N : \mathit{Public}}$$

$$\text{(ENCRYPT } \mathit{Public}\text{) } \textit{with } T = \mathit{Public} \textit{ if } k = 0\ \frac{E \vdash M_i : T \ \ i = 1..k \quad E \vdash N : \mathit{Public}}{E \vdash \{M_1, \ldots, M_k, n\}_N : T}$$

Rules (ZERO) and (SUCC) say that 0 is of level *Public* and that adding one preserves the level of a piece of data. Therefore, these classifications mean that the typing system works even against an attacker that may generate any number, starting from 0 and successively incrementing it. The rule (ENCRYPT *Public*) says that k pieces of data of the same level T can be encrypted under a key of level *Public*, with a resulting ciphertext of level T. The rule (ENCRYPT *Secret*) imposes more restrictions for encryption under keys of level *Secret*, because the resulting ciphertext is of level *Public*. These restrictions enforce a particular format for the contents and the use of a confounder: the ciphertext must contain a first component of level *Secret*, a second one of level *Any*, a third one of level *Public*, and an appropriate confounder as final component. Note that there is no rule for encryption for the case where N is a term of level *Any*.

Finally, typing rules for processes are collected in Table 25.

The first four rules handle input and output processes. Rule (OUTPUT *Public*) says that terms of level *Public* may be sent on a channel of level *Public*. Rule (OUTPUT *Secret*) says that terms of all levels may be sent on a channel of level *Secret*, provided this is done according to the correct format of a secret message. The two rules for input match these rules for output. Note that if M is a term of level *Any* and it is not known whether it is of level *Public* or *Secret*, then M cannot be used as a channel. The rule (PAIR SPLIT) breaks a term of level *Public* or *Secret* into two components, each assumed to be of the same level of the original term. The case where the origi-

Table 25 Typing rules for processes

$$\frac{E \vdash M : Public \quad E \vdash M_i : Public \;\; i = 1..k \quad E \vdash P}{E \vdash \overline{M}\langle M_1, \ldots, M_k \rangle.P} \;(\textsc{Output}\ Public)$$

$$\frac{\vdash E}{E \vdash \mathbf{0}} \;(\textsc{Dead})$$

$$\frac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q} \;(\textsc{Par})$$

$$\frac{E \vdash M : Secret \quad E \vdash P \qquad E \vdash M_1 : Secret \; E \vdash M_2 : Any \; E \vdash M_3 : Public}{E \vdash \overline{M}\langle M_1, M_2, M_3 \rangle.P} \;(\textsc{Output}\ Secret)$$

$$\frac{E \vdash P}{E \vdash !P} \;(\textsc{Repl})$$

$$\frac{E, n : T :: L \vdash P}{E \vdash (\nu n)P} \;(\textsc{New})$$

$$\frac{E \vdash M : Secret \quad E, x_1 : Secret, x_2 : Any, x_3 : Public \vdash P}{E \vdash M(x_1, x_2, x_3).P} \;(\textsc{Input}\ Secret)$$

$$\frac{E \vdash M : Public \quad E, x_i : Public \vdash P \quad i = 1..k}{E \vdash M(x_1, \ldots, x_k).P} \;(\textsc{Input}\ Public)$$

$$\frac{E \vdash M : T \quad E, x : T, y : T \vdash P}{E \vdash let\ (x, y) = M\ in\ P} \;(\textsc{Pair Split})\ T \in \{Public, Secret\}$$

$$\frac{E \vdash M : T \quad E \vdash P \quad E, x : T \vdash Q}{E \vdash case\ M\ of\ 0 : P\ succ(x) : Q} \;(\textsc{Integer})\ T \in \{Public, Secret\}$$

$$\frac{E \vdash M : T \quad E \vdash N : S \quad E \vdash P}{E \vdash [M\ is\ N]P} \;(\textsc{Match})\ T, S \in \{Public, Secret\}$$

$$\frac{E \vdash L : T \quad E \vdash N : Public \quad E, x_i : T \vdash P \;\; i = 1..k}{E \vdash case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P} \;(\textsc{Decrypt}\ Public)\ T \in \{Public, Secret\}$$

$$\frac{E \vdash L : T \quad E \vdash N : Secret \quad E, x_1 : Secret, x_2 : Any, x_3 : Public, x_4 : Any \vdash P}{E \vdash case\ L\ of\ \{x_1, x_2, x_3, x_4\}_N\ in\ P} \;(\textsc{Decrypt}\ Secret)\ T \in \{Public, Secret\}$$

nal term is known only to be of level *Any* is disallowed; if it were allowed, this rule would permit leaking whether the term is in fact a pair. The same holds true for rules (MATCH),(INTEGER) and (DECRYPT). Rule (DECRYPT *Secret*) gives the level *Any* to the confounder in the message being decrypted, for lack of more accurate static information but with no significant loss. Finally, note that there is no rule for decryption with a key of level *Any.*

Properties of the Type System. The main property of the previous type system is that if a process P typechecks, then it does not leak the values of parameters of level *Any.*

The secrecy property of well typed processes is formalized in the following theorem, where the notion of leaking is expressed via testing equivalence.

Theorem 10 (Secrecy). *If only variables of level Any and only names of level Public are in the domain of the environment E, if σ and σ' are two substitutions of values for the variables in E, and if P typechecks, i.e. $E \vdash P$, then $P\sigma$ and $P\sigma'$ are testing equivalent, i.e. $P\sigma \simeq P\sigma'$.*

The conclusion of theorem 10 means that an observer cannot distinguish $P\sigma$ and $P\sigma'$, so it cannot detect the difference in the values for the variables. Despite their secrecy, none of these variables is declared with level *Secret* ; however, the process P may produce terms of level *Secret* during its execution using the restriction operator (e.g. it may construct fresh encryption keys). For instance, P may be the process $(\nu K)(\nu m)(\nu n)\overline{c}\langle\{m,x,0,n\}_K\rangle$ where x is of level *Any* and c is of level *Public*, and where we can assign the type *Secret* to the bound names K,m,n. Theorem 10 implies that P does not leak the value x, in the sense that $P\{x := M\}$ and $P\{x := N\}$ are testing equivalent for all closed terms M and N. Thus, the typing system is meant to protect parameters of level *Any* relying on dynamically generated names of level *Secret.*

7.3 An Example with Key Establishment

We argued that the spi calculus enables more detailed descriptions of security protocols than the pi calculus. While the pi calculus enables the representation of channels, the spi calculus also enables the representation of channel implementations in terms of cryptography.

As in the pi calculus, scoping is the basis of security in spi calculus. In particular, restriction can be used to model the creation of fresh, unguessable cryptographic keys. Restriction can also be used to model the creation of fresh nonces of the sort used in challenge-response exchanges.

In this section we refine the example shown in Section 1, where we presented an abstract and simplified version of the Wide Mouthed Frog protocol. The following example is the cryptographic version of that of Section 1. In this protocol, the principals A and B share keys K_{AS} and K_{SB} respectively with a server S. When A and B want to communicate securely, A creates a new key K_{AB}, sends it to the server under K_{AS}, and the server forwards it to B under K_{SB}. Since all communication is protected by encryption, communication can take place through public channels, which we write c_{AS}, c_{SB} and c_{AB} as in Section 1. In addition to the keys and the payload M, the protocol messages include the names of principals and confounders. Informally, a simplified version of this protocol is:

Message 1: $A \rightarrow S\ \{K_{AB},*,(A,B),C_A\}_{K_{AS}}$ on c_{AS}

Message 2: $S \rightarrow B\ \{K_{AB},*,(A,B),C_S\}_{K_{SB}}$ on c_{SB}

Message 3: $A \rightarrow B\ \{*,M,*,C'_A\}_{K_{AB}}$ on c_{AB}

The channels c_{AS}, c_{BS}, c_{AB} are public. The keys K_{AS}, K_{SB} are secret keys for communication with the server, while K_{AB} is the new secret key for communication from A to B. C_A, C'_A, C_S are confounders, and $*$ is an arbitrary message of appropriate level. In Message 1, A provides the key K_{AB} to S, which passes it on to B in Message 2. In Message 1 and Message 2, the pair (A,B) conveys the names of the users of the key. In Message 3, A uses K_{AB} for sending M.

In the spi calculus, we can express this message sequence as follows, where we assume that B, after receiving the message M from A, outputs an arbitrary message on a public cannel d:

$$A(M) \triangleq (\nu K_{AB})(\nu C_A)\overline{c_{AS}}\langle\{K_{AB},*,(a,b),C_A\}_{K_{AS}}\rangle.(\nu C'_A)\overline{c_{AB}}\langle\{*,M,*,C'_A\}_{K_{AB}}\rangle$$

$$S \triangleq c_{AS}(x).case\ x\ of\ \{x_{key},x_1,x_2,x_{cnf}\}_{K_{AS}}\ in\ (\nu C_S)\overline{c_{SB}}\langle\{x_{key},x_1,x_2,C_S\}_{K_{SB}}\rangle$$

$$B \triangleq c_{BS}(x).case\ x\ of\{x_{key},x_1,x_2,y_{cnf}\}_{K_{SB}}\ in\ c_{AB}(z).case\ z\ of\{z_1,z_{cipher},z_2,z_{cfn}\}_{x_{key}}\ in\ \overline{d}\langle*\rangle\}$$

$$Inst(M) \triangleq (\nu K_{AS})(\nu K_{SB})(A(M) \mid S \mid B)$$

Now, assuming that M is a term of type *Any*, and $c_{AS}, c_{BS}, c_{AB}, d$ are channels of type *Public*, it is easy to prove that the process $Inst(M)$ is well typed. As a consequence of the theorem 10, we have that the protocol above does not reveal the message M from A. In particular, we have $Inst(M') \simeq Inst(M'')$ for arbitrary terms M', M''.

Notice that also in this version of the Wide Mouthed Frog protocol, the use of scope extrusion is essential: A generates the key K_{AB} and sends it out of scope to B via S.
In the example discussed so far, channel establishment and data communication happen only once. More sophisticated examples may be written to represent many protocol sessions between many principals. However, as the intricacy of the examples increases, so does the opportunity for errors. Note that many of the mistakes in authentication protocols arise from confusion between sessions. See [6] for further examples.

7.4 Secrecy Types for Asymmetric Communication

Although so far we have discussed only shared-key cryptography, other kinds of cryptography are also easy to treat within the spi calculus. Many security protocols use asymmetric communication primitives, namely communication channels with only one fixed end-point (the receiver) and particularly public-key encryption. Compared to shared-key encryption, these primitives present special difficulties, partly because they rely on pairs of related capabilities (e.g. "public" and "private" keys) with different level of secrecy and scopes.

In this section, we show a variant of spi calculus that focus on asymmetric communication primitives, especially public-key encryption. This process calculus has been proposed by Abadi and Blanchet in [3], where authors also show a type system in which types convey secrecy properties and such that well typed programs keep their secrets.

We consider a polyadic, asynchronous, variant of spi calculus that includes channels with only one fixed end-point (the receiver) and public-key encryption. Channels with fixed receivers can be used for transmitting secrets if the adversary cannot listen on those channels. On the other hand, the capability for sending on those channels may be published. Such channels may therefore convey not only secrets but also public data from the adversary. The type system will handle both cases.

In addition, in a public-key encryption scheme, the capabilities of encryption and decryption are separate, and can be handled separately. Typically, the capability for decryption (the "private" key) remains with one principal, while the capability for encryption (the "public" key) may be published. Our process calculus and type system treat public-key encryption and communication on channels with fixed receivers analogously.

Table 26 Syntax of the process calculus

Expressions	L,M,N	$::= a,\ldots,p,k$	name
		$\mid x,\ldots,z$	variable
		$\mid \{M_1,\ldots,M_k\}_N$	encryption ($k \geq 0$)
Processes	P,Q,R	$::= \mathbf{0}$	stop
		$\mid \overline{M}\langle N_1,\ldots,N_k\rangle$	output ($k \geq 0$)
		$\mid a(x_1,\ldots,x_k).P$	input ($k \geq 0$)
		$\mid (\nu a)P$	restriction
		$\mid P \mid P$	composition
		$\mid !P$	replication
		$\mid case\ M\ of\ \{x_1,\ldots,x_n\}_k : P\ else\ Q$	decryption ($n \geq 0$)
		$\mid if\ M = N\ then P\ else\ Q$	conditional

The syntax of the process calculus is shown in Table 26. In order to deal with asymmetric communication, Abadi and Blanchet in [3] propose to follow the same approach of the local pi calculus [40].

In the local pi calculus, input is possible only on channels that are syntactically represented by names (and not variables). Output is possible on channels represented by names or variables. Thus, the input capability for a channel a remains within the scope of the restriction $(\nu a)P$ where a is created, while the output capability can be transmitted outside. Further, this approach is extended to public-key encryption, as follows. Decryption is possible only with keys that are syntactically represented by names (and not variables). Encryption is possible with keys that are represented by names or variables. Thus we have a model where the encryption capability may be public while the decryption capability remains private, in the scope where it is generated.

Thus, when a name a refers to a channel, it represents both end-points of the channels, that is the capabilities for output and input on the channel. A variable can confer only the former capability, even if its run-time value is a. Similarly, a name k will not represent a single encryption or decryption key, but rather the pair of an encryption key and the corresponding decryption key. A variable can confer only the capability of encrypting, even if its value is k at run-time.

As an example, consider the following process:

$$(\nu k)(\overline{a}\langle k\rangle \mid b(x).case\ x\ of\ \{y\}_k : \overline{c}\langle y\rangle)$$

This process relies on three public channels, a,b,c. It generates a fresh key pair k; outputs the corresponding encryption key on a; and receives messages on b, filtering for one encrypted under k, of which it outputs the plaintext on c.

The operational semantics of the calculus can be defined in a standard way using a reduction relation and a structural congruence relation, see [3] for details.

Secrecy by Typing. In the following we show a type system such that well typed processes are proven to keep their secrets. In particular, we use a concept of secrecy similar to that we discussed for the spi calculus and in Section 4 for the pi calculus. We say that a process preserves the secrecy of a piece of data M if the process never publishes M, or anything that would permit the computation of M, even in interaction with an attacker. Moreover, we think of an attacker as any process Q of the calculus, represented by the sets of its initial capabilities (i.e. the set of names on which it is able to output, input, encrypt, and decrypt).

The types of our type system are defined by the following grammar:

$$\begin{aligned}\text{Types} ::= \mathit{Public} \mid \mathit{Secret} &\mid C^{\mathit{Public}}[T_1,\ldots,T_n] \mid C^{\mathit{Secret}}[T_1,\ldots,T_n] \\ &\mid K^{\mathit{Public}}[T_1,\ldots,T_n] \mid K^{\mathit{Secret}}[T_1,\ldots,T_n]\end{aligned}$$

Let L range over $\{\mathit{Public},\mathit{Secret}\}$, we will write $C^L[T_1,\ldots,T_n]$. We have a subtyping relation that is the least reflexive relation such that $C^L[T_1,\ldots,T_n] \leq L$ and $K^L[T_1,\ldots,T_n] \leq L$. Note that we do not have $\mathit{Secret} \leq \mathit{Public}$ or vice versa.

Public (resp. *Secret*) is the type of public (resp. secret) data. $C^{\mathit{Secret}}[T_1,\ldots,T_n]$ is the type of a channel on which the opponent cannot send messages, and which carries n-tuples with components of types T_i. Similarly, $K^{\mathit{Secret}}[T_1,\ldots,T_n]$ is the type of an encryption key that the adversary does not have, and which is used to encrypt n-tuples with components of types T_i. $C^{\mathit{Public}}[T_1,\ldots,T_n]$ is the type of a channel on which the opponent may send messages. The channel may be intended to carry n-tuples with components of types T_i, but the adversary may send *any* data it has (that is, any public data) along that channel. Similarly, $K^{\mathit{Public}}[T_1,\ldots,T_n]$ is the type of an encryption key that the opponent may have. This key is intended for encrypting n-tuples with components of types T_i, but the adversary may encrypt *any* data it has (that is, any public data) under this key.

We do not show the typing rules for this process calculus (see [3]), we only discuss the rationale of the type system.

- Any public data can be sent on a channel of type $C^{\mathit{Public}}[T_1,\ldots,T_n]$ or *Public*. This use of the channel may not seem to conform to its declared type. However, it is unavoidable, since we expect that an attacker can use the channel; moreover, it does not cause harm from the point of view of secrecy.
- Since channels of type $C^{\mathit{Secret}}[T_1,\ldots,T_n]$ may not be known by an attacker, we can guarantee that only tuples with types $T_1,\ldots,T_n$ can be sent on such a channel.
- When typing the process $a(x_1,..,x_n).P$ where a is a channel of type $C^{\mathit{Public}}[T_1,..,T_n]$, two cases arise. In the first case input values are of type *Public*; in the second case input values have the expected types $T_1,\ldots,T_n$. In order to typecheck the process $a(x_1,\ldots,x_n).P$, the type system thus checks that the process P executed after the input is well typed in both cases.
- When reading from a channel a of type $C^{\mathit{Secret}}[T_1,\ldots,T_n]$, the input values must be of the expected types $T_1,\ldots,T_n$ since the channel a cannot be known to the attacker.

- Rules for encryption are similar to those for output. Any public data can be encrypted under a public encryption key, and data of types $T_1, \ldots, T_n$ can be encrypted under a key of type $K^L[T_1, \ldots, T_n]$. Dually, rules for decryption are similar to those for input.
- Ciphertexts are always of type *Public*.

This type system reflects a binary view of secrecy, according to which the world is divided into system and attacker, and a secret is something that the attacker does not have. When we wish to express that a piece of data is a secret for a given set of principals, we define the system to include only the processes that represent those principals. Note that the mechanism of group creation we discussed in Section 4, directly supports a rich view of secrecy that does not simply divide the world in two parts. Even if that approach does not treat cryptography, we think that the type system with group creation can be extended to deal also with cryptographic primitives.

Properties of the Type System. We start with a lemma that says that every process is well-typed, at least in a fairly trivial way that makes its free names public. This lemma is important because it means that any process that represents an opponent is well-typed. It is a formal counterpart to the informal idea that the type system cannot constrain the adversary.

Lemma 2. *Let P be an untyped process. If $fn(P) \subseteq \{a_1, \ldots, a_n\}$, $fv(P) \subseteq \{x_1, \ldots, x_m\}$, and $T_i \leq Public$ for all $i = 1 \ldots m$, then $a_1 : Public, \ldots, a_n : Public, x_1 : T_1, \ldots, x_m : T_m \vdash P$.*

We end with an informal statement of the secrecy theorem, see [3] for a complete formalization.

Theorem 11 (Secrecy). *Let P be a well-typed, closed process. Then P preserves the secrecy of names of type Secret against adversaries that can input, output, encrypt, and decrypt on names declared Public, and output and encrypt on names declared $C^{Public}[\ldots]$ and $K^{Public}[\ldots]$.*

As an example, we can obtain $a{:}Public, s{:}Secret \vdash (\nu k)\overline{a}\langle\{s\}_k, k\rangle$ by letting $k : K^{Public}[Secret]$. Then the theorem above implies that the process $(\nu k)\overline{a}\langle\{s\}_k, k\rangle$ preserves the secrecy of s form any opponent that can input, output, encrypt, and decrypt on a. In other words, if Q is a closed process and $fn(Q) \subseteq \{a\}$, then $Q \mid (\nu k)\overline{a}\langle\{s\}_k, k\rangle$ does not output s on a. Thus, assuming that Q does not have s in advance, Q cannot guess s or compute it from the message on a.

7.5 Further Reading

In [6], a final section shows how we could add to the syntax of pure spi calculus cryptographic operations such as hashing, public-key encryption and digital signature.

A more general approach is that of [4], where authors introduce and study the so called *applied pi calculus*, a uniform extension of the pi calculus that is parameterized on a finite set of function symbols. Such functions can be instantiated as data structures (e.g. pairs) but also as cryptographic functions as hashing, (a)symmetric encryption, probabilistic encryption, message authentication codes (MACs). The main advantage

of applied pi calculus is that its semantics and proof techniques represent a common framework to reason about very different security protocols.

Beside secrecy, other security properties can be studied in the context of spi calculus. As an example, see [6] for a formalization of authenticity property with testing equivalence.

Finally, in [8, 7, 9] authors study the security properties of the join calculus (a variant of pi calculus with an emphasis on distributed programming [28]) enriched with cryptography.

References

1. M. Abadi. Secrecy by typing security protocols. *Journal of the ACM*, 46(5):749–786, 1999.
2. M. Abadi. Security protocols and specifications. In *Proceedings of FOSSACS'96*, pages 1–13. Springer-Verlag, LNCS 1578, 1999.
3. M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. In *Proceedings of FOSSACS'01*, pages 25–41. Springer-Verlag, 2001.
4. M. Abadi and C. Fournet. Mobile values, new names and secure communication. In *Proceedings of POPL'01*, pages 104–115. ACM Press, 2001.
5. M. Abadi and A. D. Gordon. A calculus of cryptographic protocols: the spi calculus. In *Fourth ACM Conference on Computer and Communication Security*, pages 36–47, 1997.
6. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: the spi calculus. Technical Report 149, Digital Equipment Corporation Systems Reasearch Center, January 1998.
7. Martín Abadi, Cédric Fournet, and Georges Gonthier. Secure implementation of channel abstractions. In *Proceedings of LICS '98*. IEEE, Computer Society Press, July 1998.
8. Martín Abadi, Cédric Fournet, and Georges Gonthier. A top-down look at a secure message. In C. Pandu Rangan, V. Raman, and R. Ramanujam, editors, *Proceedings of FSTTCS '99*, volume 1738 of *LNCS*, pages 122–141. Springer, December 1999.
9. Martín Abadi, Cédric Fournet, and Georges Gonthier. Authentication primitives and their compilation. In *Proceedings of POPL '00*, pages 302–315. ACM, January 2000.
10. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation*, 148(1):1–70, 1999.
11. M. Boreale, C. Fournet, and C. Laneve. Bisimulations for the join-calculus. In David Gries and Willem-Paul de Roever, editors, *Proceedings of PROCOMET '98*. IFIP, 1998.
12. Gérard Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.
13. Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Secrecy and group creation. In Catuscia Palamidessi, editor, *Proceedings of CONCUR 2000*, volume 1877 of *LNCS*, pages 365–379. Springer, August 2000.
14. Luc Maranget Cédric Fournet, Cosimo Laneve and Didier Rmy. Implicit typing à la ML for the join-calculus. In *Proc. of the 1997 8th International Conference on Concurrency Theory*. Springer, 1997.
15. Sylvain Conchon. *Analyse modulaire de flot d'information pour les calculs sąuentiels et concurrents*. PhD thesis, École Polytechnique, 2002.
16. Sylvain Conchon and Fabrice Le Fessant. Jocaml: Mobile agents for Objective-Caml. In *First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents (ASA/MA'99)*, Palm Springs, California, 1999.
17. Sylvain Conchon and François Pottier. JOIN(X): Constraint-based type inference for the join-calculus. In David Sands, editor, *Proceedings of the 10th European Symposium on Programming (ESOP'01)*, volume 2028 of *Lecture Notes in Computer Science*, pages 221–236. Springer Verlag, April 2001.

18. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
19. D.E. Denning and P.J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20:504–513, 1977.
20. 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.
21. A. Durante, R. Focardi, and R. Gorrieri. CVS: A compiler for the analysis of cryptographic protocols. In *Proceedings of CSFW'99*, pages 203–212. IEEE press, 1999.
22. A. Durante, R. Focardi, and R. Gorrieri. CVS at Work: A report on new failures upon some cryptographic protocol. In *Proceedings of MMM-ACNS'01*. Springer LNCS 2052, 2001.
23. R. Focardi, A. Ghelli, and R. Gorrieri. Using non interference for the analysis of security protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.
24. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1995.
25. R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, September 1997.
26. R. Focardi, R. Gorrieri, and F. Martinelli. "Non Interference for the Analysis of Cryptographic Protocols". In *Proceedings of ICALP'00*, number 1853 in LNCS, pages 744–755, July 2000.
27. R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proceedings of World Congress on Formal Methods (FM'99)*, pages 794–813. Springer, LNCS 1708, 1999.
28. C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, Pisa, Italy, August 26-29 1996. Springer.
29. Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*, pages 372–385, St. Petersburg Beach, Florida, January 21-24 1996. ACM.
30. Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *7th International Conference on Concurrency Theory (CONCUR'96)*, pages 406–421, Pisa, Italy, August 26-29 1996. Springer-Verlag. LNCS 1119.
31. Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *7th International Conference on Concurrency Theory (CONCUR'96)*, pages 406–421, Pisa, Italy, August 26-29 1996. Springer-Verlag. LNCS 1119.
32. J.A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of Symposium on Secrecy and Privacy*, pages 11–20. IEEE Computer Society, April 1982.
33. M. Hennessy and J. Riely. Information flow vs resource access in the asynchronous π-calculus (extended abstract). In *Proceedings of ICALP2000*, volume 1853 of *LNCS*, pages 415–427. Springer-Verlag, 2000.
34. Matthew Hennessy and James Riely. Information flow vs. resource access in the asynchronous pi-calculus. Computer Science Technical Report 2000:03, School of Cognitive and Computing Sciences, University of Sussex, 2000.
35. K. Honda, V.T. Vasconcelos, and N. Yoshida. Secure information flow as typed process behaviour. In Gert Smolka, editor, *Proceedings of ESOP 2000*, volume 1782 of *LNCS*, pages 180–199. Springer, 2000.
36. Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In P[ièrre] America, editor, *Proceedings of ECOOP '91*, volume 512 of *LNCS*, pages 133–147. Springer, July 1991.

37. A. Igarashi and Kobayashi N. A generic type system for the π-calculus. In *Proceedings of POPL '01*, pages 128–141. ACM Press, 2000.
38. G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 146–166, Passau (Germany), March 1996. Springer-Verlag, LNCS 1055.
39. W. Marrero, E. Clarke, and S. Jha. A model checker for authentication protocols. In *Proc. of DIMACS Workshop on Design and Formal Verification of Security Protocols*. Rutgers University, Sep. 1997.
40. Massimo Merro and Davide Sangiorgi. On asynchrony in name-passing calculi. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Proceedings of ICALP '98*, volume 1443 of *LNCS*, pages 856–867. Springer, July 1998.
41. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
42. R. Milner. Sorts in the π-calculus (extended abstract). In E[ike] Best and G[rzegorz] Rozenberg, editors, *Proceedings of the 3rd Workshop on Concurrency and Compositionality*, volume 191 of *GMD-Studien*. GMD Bonn, St. Augustin, 1991. Also available as Report 6/91 from University of Hildesheim.
43. R. Milner. The polyadic π-calculus: A tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.
44. R. Milner. The π-calculus. Undergraduate lecture notes, Cambridge University, 1995.
45. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, May 1999.
46. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, September 1992.
47. B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Compoter Sciences*, 6(5):409–454, 1996. An extended abstract in *Proc. LICS 93*, IEEE Computer Society Press.
48. R. Focardi and F. Martinelli. A uniform approach for the analysis of cryptographic protocols. In *Conference on Security in Communication Networks (SCN'99), (G. Persiano ed.)*, 1999.
49. R. Focardi, R. Gorrieri, and F. Martinelli. Secrecy in security protocols as non interference. In *Proceedings of DERA/RHUL Workshop on Secure Architectures and Information Flow (S. Schneider and P. Ryan ed.)*. Elsevier ENTCS, Volume 32, 1999.
50. R. Focardi, R. Gorrieri, and F. Martinelli. Message authentication through non interference. In *Proceedings of International Conference on Algebraic Methodology And Software Technology (AMAST 2000)*, pages 258–272. Springer LNCS 1816, 2000.
51. A. W. Roscoe, J. C. P. Woodcock, and L. Wulf. Non-interference through determinism. In *Proceeding of European Symposium on Research in Computer Security 1994 (ESORICS'94)*, number 875 in LNCS, pages 33–53. Springer-Verlag, 1994.
52. P.Y.A. Ryan and S.A. Schneider. Process algebra and non-interference. In *Proceedings of 12th IEEE Computer Security Foundation Workshop*, pages 214–227. IEEE press, 1999.
53. A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundation Workshop*, Cambridge (UK), 2000. IEEE press.
54. D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
55. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings of POPL'98*, pages 355–364. ACM Press, January 1988.
56. D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

Classification of Security Properties*
(Part II: Network Security)

Riccardo Focardi[1], Roberto Gorrieri[2], and Fabio Martinelli[3]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy
focardi@dsi.unive.it
[2] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy
gorrieri@cs.unibo.it
[3] Istituto di Informatica e Telematica, C.N.R., Pisa, Italy
Fabio.Martinelli@iit.cnr.it

Abstract. Many security properties of cryptographic protocols can be all formalized as specific instances of a general scheme, called Generalized Non Deducibility on Composition (*GNDC*). This scheme derives from the *NDC* property we proposed a few years ago for studying information flow in computer systems. The theory is formulated for CryptoSPA, a process algebra we introduced for the specification of cryptographic protocols. One of the advantages of our unifying *GNDC*-based theory is that that formal comparison among security properties become easier, being them all instances of a unique general property. Moreover, the full generality of the approach has helped us in finding a few undocumented attacks on cryptographic protocols.
This paper is based on the results of [20,22,23,24,25] and covers the second part of the course "Classification of Security Properties" given by Roberto Gorrieri and Riccardo Focardi at the FOSAD'00 and FOSAD'01 schools.

1 Introduction

Many security properties of cryptographic protocols have been identified in recent years, such as *secrecy* (confidential information should be available only to the partners of the communication), *authentication* (capability of identifying the other partner engaged in a communication), *integrity* (assurance of no alteration of message content), *non repudiation* (assurance that a signed document cannot be repudiated by the signer), *fairness* (in a contract, no party can obtain advantage by ending the protocol first), and some others.

Even if there is a widespread agreement on what is the intended meaning of these properties, under a closer scrutiny one realizes that they are very slippery properties, especially authentication. As a matter of fact, formal definitions of,

* Work partially supported by the EU Contract IST-2001-32617 MyThs, the MIUR project "Formal Methods for Security" (MEFISTO) and Microsoft Research Europe. The third author is also supported by the project "Model based design/validation for Web Services (MbDWS)".

R. Focardi and R. Gorrieri (Eds.): FOSAD 2001/2002, LNCS 2946, pp. 139–185, 2004.

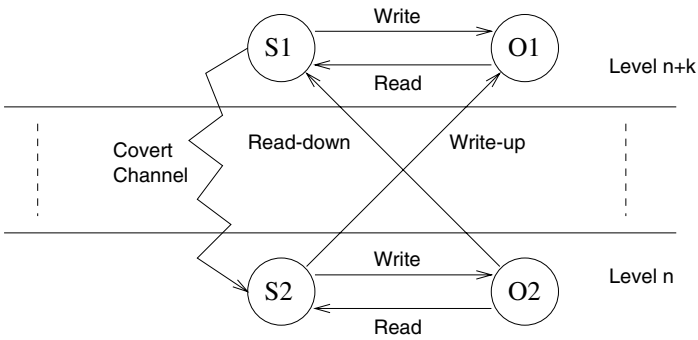


Fig. 1. Multilevel security: a high subject S_1 cannot write a low object O_2 and a low subject S_2 cannot read a high object O_1.

e.g. authentication, have been given rarely, not widely agreed upon, usually not compared and only recently proposed in the literature (see, e.g., [5,27,33,42]). This is sometimes due to the fact that we first need a formal model on which the problem is defined (and this is often a source of possible proliferation of different proposals) and then a formal definition with respect to the chosen model. Moreover, even when a formal definition is given, usually this is not (easily) comparable to others, due to different mathematical assumptions of the model.

Our claim is that a classic approach to security, used to study information flow in multilevel [6] computer systems, can be profitably used also for the analysis of security properties in network protocols.

1.1 Multilevel Security and Non-interference

In a multilevel systems, processes/users and objects are bound to a specific security level (e.g., in the military jargon, unclassified, classified, secret and top secret) and information can only flow from low levels to higher ones. This is usually implemented by constraining the possible actions of processes according to the rules of *no read-up* and *no write-down* (see Fig. 1).

The advantage of this approach with respect to conventional approaches used in commercially available operating systems (e.g., Unix) is that the possible information disclosures caused by the inadvertent execution of a Trojan Horse program is confined inside the level of the user that executed it. However, these two rules are not enough as indirect information flows, usually called *covert channels*, may be possible when using some shared resource. For instance, it is not difficult to build a Trojan Horse program that, once executed by a high level user, is able to downgrade information by synchronizing with a low level process on the system side-effects generated by repeatedly filling the shared hard disk: the high process can transmit a bit 0 by causing a disk-full error on the low level attempt to write, or a bit 1 by allowing the low process to write.

To solve the problem of preventing unauthorized information flows, be they direct or indirect, in the last two decades many proposals have been presented,

starting from the seminal idea of *non interference* proposed in [26] for deterministic state machines. In recent work [17,18,19], two of the authors have studied the many non interference-like definitions in the literature, by defining all of them uniformly in a common process algebraic setting, producing the first taxonomy of these security properties reported in the literature.

In [17,18,19], we use a CCS-like process algebra [38], called Security Process Algebra (SPA for short), where the set of actions is partitioned into two sets L and H of low actions and high ones, respectively. Processes built by using only actions in H (L) are called high (low) level processes. These processes are secure by construction, because their activities (expressed by the actions they perform) are confined inside the high (low) level. More interesting is the case of *mixed* (built with actions from both L and H) processes, because only for them information can flow between the two levels. Of course, we are interested in detecting if information flow in the wrong direction (from high to low). Among the many non interference-like properties, we advocate a particular one, called *Non Deducibility on Composition* (*NDC* for short), that can be expressed as follows:

$$P \in NDC \text{ iff } \forall \Pi \in \mathcal{E}_H : (P \parallel \Pi)\backslash H \approx P\backslash H$$

where $\mathcal{E}_H$ is the set of all high level processes, $\approx$ is a behavioural equivalence[1] relation, $\parallel$ is the CCS parallel composition and $\backslash$ is the CCS restriction operator. Hence, on the one hand, $P\backslash H$ is able to exhibit only the low level behaviour of P, while $(P \parallel \Pi)\backslash H$ is the low level behaviour of $P \parallel \Pi$. The basic intuition is that the requirement of

No information flow from high to low

is expressed by the extensional, behavioural condition

No high level process can change the low behaviour.

1.2 Non-interference for Security Protocols

When considering network security, other kinds of problems become relevant. In particular, communication protocols are executed on unreliable networks (hence, messages can be lost or duplicated) which are also insecure (hence, messages can be intercepted, fabricated and possibly modified by third, malicious parties that can control the network traffic). In order to achieve reasonable security services (e.g., secrecy of the transmitted data), network protocols typically exploits cryptographic primitives. Such enhanced protocols are usually called *cryptographic protocols*.

In the analysis of cryptographic protocols, one has to cope with the insecurity of the network. So, it is assumed that an external, possibly malicious, *attacker* (sometimes called *enemy* or *intruder*) of the protocol has complete control over the communication medium. On the other hand, it is also usually assumed *perfect cryptography*, i.e., such an enemy is not able to perform cryptanalytic attacks:

[1] Actually, *NDC* in [17,18] is this property when $\approx$ is trace equivalence; variations on the theme are obtained by changing the relation, e.g., $BNDC$ is as above where $\approx$ is weak bisimulation, as we will see in the sequel.

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 [12].

In order to model cryptographic protocols, the SPA process algebra is enriched with primitives for handling messages and cryptographic functions. The resulting process algebra is called CryptoSPA [25].

Correspondingly, the *NDC*-based analysis technique for security in computer systems has to be adapted. *NDC* 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. On the other hand, 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, the low level actions 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. For instance, we will see that to model some form of authentication as in [32], it is enough to include special start/commit actions for all the honest participants.

Furthermore, enemies 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 information, such as names of entities and public keys, and its own private data (e.g., enemy's private key). Hence, by following [25,24,22], the set $\mathcal{E}_C^{\phi_I}$ of all the admissible attackers is as follows: $\mathcal{E}_C^{\phi_I} = \{X \mid sort(X) \subseteq C \text{ and } ID(X) \subseteq \mathcal{D}(\phi_I)\}$, where C is the set of public channel names, $sort(X)$ is the set of channel names syntactically occurring in X, $ID(X)$ is the set of messages that syntactically appear in X, ϕ_I is the initial knowledge given to any enemy X, and $\mathcal{D}$ is a deduction system that manipulates (blocks of) messages in the obvious way (e.g., a crypted information can be disclosed if the decryption key is known). By requiring that all the messages in $ID(X)$ are deducible from ϕ_I we are stating that the enemy cannot know in advance messages that are not derivable by the explicitly given set ϕ_I. The *NDC* property for a protocol P can hence be reformulated as:

$$P \in NDC_C^{\Phi_I} \text{ iff } \forall X \in \mathcal{E}_C^{\phi_I} : (P \,\|\, X) \setminus C \approx P \setminus C$$

On the one hand, $P \setminus C$ represents the secure specification of the protocol P running in isolation on perfectly secure channels. The visible behaviour of P is given by the property dependent, extra observable actions included into the specification. Hence, the behaviour of $P \setminus C$ describes the protocol in isolation, where the security property of interest holds. On the other hand, if $P \setminus C$ is equivalent to $(P \,\|\, X) \setminus C$, then this clearly means that X is not able to modify

in any way the observable execution of P, i.e., the security property holds even when P is executed in any possible hostile environment with initial knowledge Φ_I. Intuitively, in the setting of cryptographic protocols, interferences may represent possible attacks. As a consequence, *non interference* guarantees the absence of attacks.

Interestingly enough, when the observational equivalence $\approx$ is trace equivalence (two systems are equivalent if they perform the same set of traces), then *NDC* can be characterized in a simpler way, by finding a canonical, most general enemy that can be used in place of all. By removing the universal quantification, *NDC* can be verified by one single, albeit huge, check. The most general enemy is an attacker that can eavesdrop/intercept any message (adding the intercepted information to its knowledge set), as well as produce new messages with pieces of information he knows.

1.3 The *GNDC* Scheme

The Generalized *NDC* (*GNDC*) scheme [25,23] relaxes, in some aspects, the *NDC* property presented above, in order to express uniformly many security properties. As we will see in the next sections, it is general enough to capture many different properties of cryptographic protocols. To give the flavour of how *GNDC* works, we present a very simple example of a key-exchange protocol. For such a protocol, we consider two authentication properties and we show how they can be formalized in terms of *observable events*. Finally, we show how such formalizations can be seen as instances of *GNDC*.

The simple key-exchange protocol we consider has the aim of distributing a fresh session-key K_s from *Alice* (A) to *Bob* (B) by using a long-term key K shared between such users. The new key K_s will be used by Alice and Bob to communicate inside the current session. Session-key distribution is a (standard) way to reduce the risk of cryptanalytic attacks on cryptographic keys, as it allows two parties to establish a new key for each new session.

$$A \to B : \{K_s, A\}_K$$

The notation $A \to B : msg$ above represents A sending message msg to B; moreover $\{M\}_K$ is used to denote the encryption of M with the key K.

In order for this protocol to be effective, it must intuitively provide both authentication and secrecy of the session key K_s. In particular, B should be guaranteed that K_s really comes from A and that nobody else knows it. Both of these properties should be guaranteed by the encryption of K_s with K. Here we focus on authentication and we consider two variants of it:

(i) *key authenticity* requires that key K_s is authentic from A; this should be guaranteed by the fact that only A and B know the long-term key K: when B decrypts the message he concludes that only A may have generated it. In case this property is required on a generic message M (not necessarily a key), we refer to it as *message authenticity*;

(*ii*) *entity authentication* requires that, at the end of the protocol, B is convinced that A has been running the protocol with him; again, this should be guaranteed by the fact that only A and B know the long-term key K.

In the following we show that the two properties above do not hold on the simple protocol we are considering, when multiple sessions are considered.

Message Authenticity. A simple way to check message authenticity is to fix in advance the key K_s that A is willing to distribute to B and require that in all the possible (concurrent) runs of the protocol B always receives the correct key K_s, i.e. the key that A wanted to send to B. If this is true, we can conclude that the protocol guarantees that no one is able to force B accepting a fake session key K'_s. This notion of message authenticity is due to Abadi and Gordon [2].

A first important thing to observe is that considering all the possible runs is not enough. At least we need to be more precise about what we mean by run. As a matter of fact, we have to consider all the possible executions of the protocol in every possible (potentially hostile) environment. It is certainly different if we consider the protocol execution with or without the presence of some malicious enemy which tries to send a fake K'_s. We can thus rephrase the message (key) authenticity property as follows:

> "Whatever hostile environment is considered, B will never receive (as part of Message 1), during all of his possible runs, a key which is different from K_s".

Notice that the property above looks really ad-hoc for the specific protocol we are considering. As a matter of fact, we are requiring that a particular piece of information sent inside a particular message differs from a certain fixed key K_s. This specificity is a quite typical aspect of precise definitions of security properties, since they often depend on the structure of the analyzed system or protocol. We can improve the generality of the property by assuming to have an event $received(m)$ corresponding to the reception of message m by B and by denoting with $P(k)$ the protocol specification in which *Alice* is willing to send the session-key k to *Bob*. We can now state that

> "$P(K_s)$ guarantees message (key) authenticity if whatever hostile environment is considered, an event $received(K'_s)$ with $K'_s \neq K_s$ can never occur".

We can now show that the simple protocol we have considered so far does not guarantee message authenticity, when more than one protocol session is possible. The flaw is due to a well-known *replay attack*, that allows the enemy to make B accept an old session key. This is very dangerous as the enemy could have time to break an old session key and force B using such a broken key again. Knowing the key, the enemy can completely impersonate A in the just established session. The attack sequence follows:

message 1a	$A \rightarrow B : \{K_s, A\}_K$	*The enemy stores this message*
...	...	...
message 1b	$\mathbf{E}(A) \rightarrow B : \{K_s, A\}_K$	*The enemy replays the old message*

$\mathbf{E}(A)$ denotes the enemy impersonating A.

To see why this attack is revealed by the definition of message authenticity we have given, it is sufficient to consider the situation in which two (sequential) instances of P are considered, with two different keys: $P(K_s^1), P(K_s^2)$. Now it is easy to see that $P(K_s^2)$ does not guarantee message authenticity. Indeed, the enemy may exploit the attack sequence above to make the B of $P(K_s^2)$ accept the session key K_s^1 exchanged in the first session $P(K_s^1)$. As a consequence B will produce the event $received(K_s^1)$ instead of the expected one $received(K_s^2)$. This may be depicted as follows:

$$\begin{array}{lll} \text{message } 1a & A \to B : \{K_s, A\}_K & received(K_s^1) \\ \ldots & \ldots & \ldots \\ \text{message } 1b & \mathbf{E}(A) \to B : \{K_s, A\}_K & received(K_s^1) \end{array}$$

Entity Authentication. We now consider another security property: entity authentication. This property is more subtle. Informally, entity authentication should allow the verification of an entity's claimed identity, by another entity. There are several attempts in the literature to formalize this notion. Here, we follow the ones based on *correspondence* between actions of the participants (see, e.g., [28,33,45]).

As an example, in our protocol we would like that whenever B receives the protocol message (encrypted with the correct key) then A has indeed executed the protocol with him. To formalize this idea, we consider two events $commit(B, A)$ and $run(A, B)$ representing the fact that B has successfully terminated the protocol apparently with A and A has at least started the protocol with B. It is now sufficient to require that each event $commit(B, A)$ is always matched by an event $run(A, B)$ [33]. In other words $commit(B, A)$ should never happen if A has not started the protocol. We state entity authentication as follows:

> "P guarantees entity authentication of B with respect to A if, whatever hostile environment is considered, an event $commit(B, A)$ can never occur without a matching event $run(A, B)$".

Note that the same attack considered for message authenticity is also an attack for entity authentication:

$$\begin{array}{lll} \text{message } 1a & A \to B : \{K_s, A\}_K & run(A, B),\ commit(B, A) \\ \ldots & \ldots & \ldots \\ \text{message } 1b & \mathbf{E}(A) \to B : \{K_s, A\}_K & commit(B, A) \end{array}$$

Notice that in the second message we have an event $commit(B, A)$ without the matching event $run(A, B)$, representing the fact that A is not running the protocol in the second session. This captures the replay attack performed by E, in which E impersonates A. We conclude that B cannot be guaranteed of the identity of the claimant, i.e., P does not guarantee entity authentication.

Remark 1. Entity authentication and message authenticity are different properties. As an example, if we do not have a message "to be sent" by the entity that

we want to authenticate, message authenticity property becomes senseless. To see this point, consider the following (faulty) authentication protocol:

$$\begin{array}{lll} \text{message 1} & A \to B : \{N_A\}_{K_{AB}} & run(B, A) \\ \text{message 2} & B \to A : N_A & commit(A, B) \end{array}$$

In order to verify the identity of B, A sends a challenge N_A (typically a random number used only once, called nonce) to B encrypted with the symmetric key K_{AB} which is only known by A and B. Only B should be able to decrypt N_A and send it back to A. If A may play both the initiator and responder role in parallel sessions of the protocol, the following well-known *reflection attack* becomes possible:

$$\begin{array}{lll} \text{message 1} & A \to \mathbf{E}(B) : \{N_A\}_{K_{AB}} & \\ \text{message 1}' & \mathbf{E}(B) \to A : \{N_A\}_{K_{AB}} & \\ \text{message 2}' & A \to \mathbf{E}(B) : N_A & \\ \text{message 2} & \mathbf{E}(B) \to A : N_A & commit(A, B) \end{array}$$

The enemy intercepts the first message and starts a new session of the protocol with A. Basically, the enemy uses this second session to obtain from A the value N_A. Finally the enemy can conclude the first session successfully masking as B. Note that, again, we have a $commit(A, B)$ event with no $run(B, A)$.

Notice that B has no private messages to send to A, thus it is not possible to express the attack above as an attack on message authenticity. Indeed, B does not send any message to A that should be authentic from him.

The General Scheme. We have seen that the two different properties of message authenticity and entity authentication, can be both specified by requiring that whatever hostile environment is considered, the protocol never shows some particular *bad behaviour*. We have also observed that this set of bad behaviours, in general, depends on the particular property and sometimes may also depend on the protocol P. For example, for message authenticity we need the parameter m of P in order to require that m is the message to be delivered. We take a complementary approach (that is more manageable), and we denote with $\alpha_S(P)$ the set of all possible good behaviour of P with respect to the security property S. Moreover we denote with $\gamma_S(P)$ the protocol P decorated with the suitable events of property S. Informally, our general scheme becomes the following:

> "P guarantees a security property S if, whatever hostile environment is considered, $\gamma_S(P)$ always shows behaviours in $\alpha_S(P)$".

The considerations above are at the base of the proposal of a uniform formal framework where security properties can be defined. The proposed schema, called Generalized *NDC* (*GNDC* for short), is as follows:

$$P \text{ is } GNDC_{\lhd}^{\gamma_S,\alpha_S} \text{ iff } \forall X \in \mathcal{E}_C^{\phi_I} : (\gamma_S(P) \parallel X) \setminus C \lhd \alpha_S(P)$$

where $\lhd$ is a behavioural preorder, i.e., $P \lhd P'$ means that all the behaviours shown by P are also shown by P'.

$GNDC_{\lhd}^{\gamma_S,\alpha_S}$ corresponds to the informal scheme sketched above. Indeed, $X \in \mathcal{E}_C^{\phi_I}$ selects enemies with a knowledge limited by ϕ_I. Then, the decorated protocol $\gamma_S(P)$ executed with X is required to show behaviours inside the set $\alpha_S(P)$ of good ones.

Given the *GNDC* scheme, we can define a specific property S by suitably instantiating the function $\alpha_S(P)$, the preorder $\lhd$ and the decoration function γ_S. We will see that many (network) security properties can be instantiated in the *GNDC* scheme. Here, we informally show how this is achieved for non interference (formalized as NDC), message authenticity and entity authentication:

- *Non-interference (NDC)*: $\alpha_{NI}(P) = P \setminus C$, $\lhd$ is $\approx$ and $\gamma_{NI}(P) = P$. We obtain the exact definition of *NDC*.
- *Message authenticity*: $\alpha_{MA}(P(M))$ is the process where event $received(M')$ may only occur with $M' = M$. $\lhd$ is the trace-preorder, i.e., $P \lhd P'$ if all the execution sequences (i.e., trace) of P are execution sequences of P' and $\gamma_{MA}(P)$ is the process that suitably performs $received(m)$, whenever the supposedly authentic message m is delivered by the protocol.
- *Entity Authentication*: $\alpha_{EA}(P)$ is the process where events $commit(B, A)$ are always preceded by matching events $run(A, B)$; $\lhd$ is the trace-preorder, as above, and $\gamma_{EA}(P)$ is the process that suitably performs $run(A, B)$ and $commit(B, A)$, whenever A starts the protocol with B and B concludes the protocol convinced to communicate with A, respectively.

1.4 Plan of the Paper

The primary goal of this paper is to substantiate our claim that most (maybe all) security properties proposed for the analysis of cryptographic protocols are expressible as suitable instances of the *GNDC* schema above, by suitably choosing the property dependent (low) observable actions, the position where they are to be inserted into the specification, as well as a suitable behavioural equivalence. One interesting result that holds for trace semantics is that, once fixed the function γ, *NDC* is the strongest property in the $GNDC^{\alpha}$ family, for any choice of the function α.

We think that the advantages of our approach include at least the following:

- *formal comparison*: as the definitions are now given in a uniform style, it should be easier to compare the relative merits; this is especially true for slippery properties such as the many varieties of authentication.
- *one check for all*: As all the properties are defined in the same *NDC* style and *NDC* is the strongest property, it is possible to put into the specification the extra actions for all the properties of interest, hence obtaining that one successful *NDC* check for this rich case implies that all the properties are satisfied.
- *accuracy*: So far we have analyzed about 40 protocols (with the help of an automatic tool [13,15]) of a well-known library of crypto-protocols [10]. Two supposedly correct protocols have been shown incorrect and for a few additional flawed protocols some new attacks have been found (see [14] for details). Our experience hence supports our claim that a protocol passing the *NDC* test is more likely to be flaw free.

The paper is organized as follows. Section 2 introduces the process algebra CryptoSPA that we use to formally define cryptographic protocols. Section 3 describes how the non interference approach, developed for SPA in [17,18,19], is rephrased over CryptoSPA. In particular, we study how to describe security properties by decorating the protocol specification with specific low level actions; we also study when an attacker (or enemy) is admissible (by imposing restriction on its initial knowledge) and, finally, the behavioural semantics for CryptoSPA we are interested in (trace, bisimulation), over which the *NDC* theory is parametrized. Section 4 addresses some technical problems we have to face in order to perform cryptographic protocol verification. For instance, the problem of finding the most general enemy, in order to remove the universal quantification in the *NDC* definition, as well as the problem of compositionality of secure cryptographic protocols. In Section 5 the general *GNDC* scheme is presented, as well as its instantiation to several security properties, including secrecy, message authenticity, entity authentication, non repudiation and fairness. Section 6 provides some formal comparison among security properties. Among the results reported there, we mention the proof that *NDC* is the strongest property in the *GNDC* family. Section 7 concludes the paper with some comparison with related work and some suggestions for future research.

2 The Model

In this section we describe the language we use for the specification of cryptographic protocols and their security properties. It is called *Cryptographic Security Process Algebra* (CryptoSPA for short), and is a variant of value-passing CCS [38], where the processes are provided with some primitives for manipulating messages [36]. In particular, processes can perform message encryption and decryption, and also construct complex messages by composing together simpler ones.

2.1 The CryptoSPA Syntax

The CryptoSPA syntax is based on the following elements:

- A set $I = \{a, b, \ldots\}$ of *input* channels and a set $O = \{\bar{a}, \bar{b}, \ldots\}$ of *output* channels, related through a function $\bar{\cdot} : I \cup O \to I \cup O$ which given an input $a \in I$ returns the *corresponding* output $\bar{a} \in O$ and vice-versa, i.e., $\bar{\bar{a}} = a$.
- A set M of basic messages. The set $\mathcal{M}$ of all messages is defined as the least set such that $M \subseteq \mathcal{M}$ and $\forall m, m', k \in \mathcal{M}$ we have that (m, m') (pairs) and $\{m\}_k$ (encryptions) also belong to $\mathcal{M}$.
- A set $C \subseteq I \cup O$ of channels, ranged over by c, such that $c \in C$ iff $\bar{c} \in C$; these *public* channels represent the insecure network on which the enemy can intercept and fake messages. Channels in $(I \cup O) \setminus C$ are the *private* channels.
- A function $Msg : I \cup O \longrightarrow \mathcal{P}(\mathcal{M})$ which maps every channel c into the set of possible messages that can be sent and received along such a channel. Msg is such that $Msg(c) = Msg(\bar{c})$.

- A set $Act = \{c(m) \mid c \in I, m \in Msg(c)\} \cup \{\overline{c}\, m \mid \overline{c} \in O, m \in Msg(c)\} \cup \{\tau\}$ of actions (τ is the internal, invisible action), ranged over by a (with abuse of notation).
- A set Var of *variables*, raged over by x.
- A set $Const$ of constants, ranged over by A.

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

$$P ::= \underline{0} \mid c(x).P \mid \overline{c}\, e.P \mid \tau.P \mid P + P \mid P \parallel P \mid P \setminus L \mid P[f] \mid \\ \mid [e = e']P; P \mid [\langle e_1 \dots e_r \rangle \vdash_{rule} x]P; P \mid A(e_1, \dots, e_n)$$

where $e, e', e_1, \dots, e_r$ are messages or variables, L is a set of input channels and $f : Act \mapsto Act$ is a function that relabels channel names inside actions[2] and may also relabel actions into the internal (invisible) action τ. Both the operators $c(x).P$ and $[\langle e_1 \dots e_r \rangle \vdash_{rule} x]P; P'$ bind the variable x in P.

Most of the CryptoSPA operators are the same as in value-passing CCS. Intuitively, $\underline{0}$ is the empty process, which cannot do any action; $c(x).P$ reads a value v from channel c and then behaves as process P in which all the free occurrences of x are replaced by v; symmetrically, $\overline{c}\, e.P$ sends e as output on channel c then behaving as P; $\tau.P$ can do an internal action τ and then behaves like P; $P_1 + P_2$ can alternatively choose[3] to behave like P_1 or P_2 ; $P_1 \parallel P_2$ is the parallel composition of P_1 and P_2, where the executions of the two systems are interleaved, possibly synchronized on complementary input/output actions, producing an internal τ; $P \setminus L$ can execute all the actions P is able to do, provided that inputs and outputs are not performed over channels belonging to L; if P can execute action a, then $P[f]$ performs $f(a)$; finally, $[e = e']P_1; P_2$ behaves like P_1 if $e = e'$ and like P_2 otherwise.

Besides the above described standard value-passing CCS operators, we have an additional one that has been introduced in order to model message handling and cryptography. Informally, the $[\langle m_1 \dots m_r \rangle \vdash_{rule} x]P_1; P_2$ process tries to deduce a piece of information z from the tuple of messages $\langle m_1 \dots m_r \rangle$ through one application of rule $\vdash_{rule}$; if it succeeds, then it behaves like $P_1[z/x]$, otherwise it behaves like P_2. See the next subsection for a more detailed explanation of derivation rules.

Finally, let Def : $Const \mapsto \mathcal{L}$ be a set of defining equations of the form $A(x_1, \dots, x_n) \stackrel{\text{def}}{=} P$, where P may contain no free variables except $x_1, \dots, x_n$, which must be distinct. Constants permit us to define recursive processes, but we have to be a bit careful in using them. A term P is *closed* with respect to Def if all the constants occurring in P are defined in Def (and, recursively, for their defining terms). A term P is *guarded* with respect to Def if all the constants occurring in P (and, recursively, for their defining terms) occur in a prefix context [38]. A term P is *finite* with respect to Def if the set of constants

[2] The relabeling functions map channels in C to channels in C and channels in $(I \cup O) \setminus C$ to channels in $(I \cup O) \setminus C$.

[3] For notational convenience, we use sometimes the $\sum$ operator (indexed on a set) to represent a general n-ary (or even infinitary) sum operator.

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

$$\frac{m \quad k}{\{m\}_k}\ (\vdash_{enc}) \qquad \frac{\{m\}_k \quad k^{-1}}{m}\ (\vdash_{dec})$$

Fig. 2. Inference System for message manipulation, where $m, m', k, k^{-1} \in \mathcal{M}$.

occurring in P or, recursively, in their defining terms is finite. We call $\mathcal{E}$ the set of all the CryptoSPA *closed* terms (i.e., with no free variables), that are closed, guarded and finite with respect to Def. Terms in $\mathcal{E}$ are usually called CryptoSPA *processes*. Let $sort(P)$ denote the *sort* of P, i.e., the set of channels occurring syntactically in P. With $\mathcal{E}_C$ we denote the processes P such that $sort(P) \subseteq C$.

For the sake of readability, we always omit the termination $\underline{0}$ at the end of process specifications, e.g., we write a in place of $a.\underline{0}$. We also write $[m = m']P$ in place of $[m = m']P;\underline{0}$ and analogously for $[\langle m_1 \dots m_r\rangle \vdash_{rule} x]P;\underline{0}$. Finally, we often replace constructive rules (encryption and pairing) with the resulting messages, e.g., we use $\overline{c}\,\{m\}_k$ as a shorthand for $[\langle m,k\rangle \vdash_{enc} x]\overline{c}\,x$.

2.2 The Operational Semantics of CryptoSPA

In order to model message handling and cryptography, CryptoSPA may be equipped with a set of suitable inference rules (inference system). Note that CryptoSPA syntax, its semantics and the results obtained herein are completely parametric with respect to the chosen inference system. For explanatory purposes, in Figure 2 we provide a simple inference system which is quite similar to those used by many authors (see, e.g., [32,34]). However, it is possible (and easy) to adopt other rules, e.g., for modeling different kinds of cryptographic approaches as well as cryptographic weaknesses. We consider a function $\cdot^{-1} : \mathcal{M} \to \mathcal{M}$ which denotes, for each key k (i.e., a message possibly used as encryption key), the corresponding decryption key. Note that there are no rules to obtain the message k^{-1} from k (and *vice-versa*). In particular the inference system can combine two messages obtaining a pair (rule $\vdash_{pair}$); it can extract one message from a pair (rules $\vdash_{fst}$ and $\vdash_{snd}$); it can encrypt a message m with a key k obtaining $\{m\}_k$ and, finally, decrypt a message of the form $\{m\}_k$ only if it has the corresponding (inverse) key k^{-1} (rules $\vdash_{enc}$ and $\vdash_{dec}$). As an example, process $[\langle\{m\}_k, k^{-1}\rangle \vdash_{dec} x]P_1;P_2$ decrypts message $\{m\}_k$ through the inverse key k^{-1} and behaves like $P_1[m/x]$, while $[\langle\{m\}_k, k'\rangle \vdash_{dec} x]P_1;P_2$ (with $k' \neq k^{-1}$) tries to decrypt the same message with the wrong inverse key k' and (since it is not permitted by $\vdash_{dec}$) it behaves like P_2.

Given an inference system, we say that a message m can be deduced from a set of messages ϕ whenever there exists a proof tree whose nodes are messages, such that the root is m, the leaves are contained in ϕ and each message in the tree may be obtained by applying a rule instance of the inference system whose

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

$$(\|_1)\frac{P \xrightarrow{a} P'}{P \,\|\, P_1 \xrightarrow{a} P' \,\|\, P_1} \qquad (\|_2)\frac{P \xrightarrow{c(m)} P' \quad P_1 \xrightarrow{\overline{c}\,m} P_1'}{P \,\|\, P_1 \xrightarrow{\tau} P' \,\|\, P_1'} \qquad (+_1)\frac{P \xrightarrow{a} P'}{P + P_1 \xrightarrow{a} P'}$$

$$(=_1)\frac{m = m' \quad P_1 \xrightarrow{a} P_1'}{[m = m']P_1; P_2 \xrightarrow{a} P_1'} \qquad (=_2)\frac{m \neq m' \quad P_2 \xrightarrow{a} P_2'}{[m = m']P_1; P_2 \xrightarrow{a} P_2'} \qquad ([f])\frac{P \xrightarrow{a} P'}{P[f] \xrightarrow{f(a)} P'[f]}$$

$$(\backslash L)\frac{P \xrightarrow{c(m)} P' \quad c \notin L}{P\backslash L \xrightarrow{c(m)} P'\backslash L} \qquad (\mathcal{D}_1)\frac{\langle m_1 \ldots m_r\rangle \vdash_{rule} m \quad P_1[m/x] \xrightarrow{a} P_1'}{[\langle m_1 \ldots m_r\rangle \vdash_{rule} x]P_1; P_2 \xrightarrow{a} P_1'}$$

$$(\mathcal{D}_2)\frac{\nexists m : \langle m_1 \ldots m_r\rangle \vdash_{rule} m \quad P_2 \xrightarrow{a} P_2'}{[\langle m_1 \ldots m_r\rangle \vdash_{rule} x]P_1; P_2 \xrightarrow{a} P_2'}$$

$$(def)\frac{P[m_1/x_1, \ldots, m_n/x_n] \xrightarrow{a} P' \quad A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P}{A(m_1, \ldots, m_n) \xrightarrow{a} P'}$$

Fig. 3. Operational semantics (symmetric rules for $+_1$, $\backslash L$, $\|_1$ and $\|_2$ are omitted).

premises are the descendants of the message in the tree. We consider a function $\mathcal{D}$, from finite sets of messages to sets of messages, such that $\mathcal{D}(\phi)$ is the set of messages that can be deduced from ϕ. We assume that $\mathcal{D}(\phi)$ is a decidable set.

Note that, in our model, we are assuming encryption as completely reliable. Thus, we do not allow any kind of cryptographic attack, e.g., the guessing of secret keys. Nevertheless, the goal of the theory we are going to present is to capture those attacks that can be carried out even if cryptography is completely reliable. The behavior of a CryptoSPA term is formally described by means of the *labeled transition system* $\langle \mathcal{E}, Act, \{\xrightarrow{a}\}_{a \in Act}\rangle$, where $\xrightarrow{a}_{a \in Act}$ is the least relation between CryptoSPA terms induced by the axioms and inference rules of Figure 3.

The behaviour of a specific term P is the portion of the above labelled transition system, which is reachable starting from the state P. We write $P \equiv P'$ if P and P' show exactly the same behaviour, i.e., if they have isomorphic labelled transition systems, up to state renaming.

Example 1. We give a simple example of CryptoSPA protocol specification. Consider again the (flawed) key exchange protocol we presented in section 1.3:

$$A \to B : \{K_s, A\}_K$$

It can be specified as the following CryptoSPA process P:

$$P = \overline{c}\,\{K_s, A\}_K.A'(K_s) \,\|\, c(y).[\langle y, K\rangle \vdash_{dec} w][w \vdash_{fst} z].B'(z)$$

where $K^{-1} = K$ (symmetric encryption) and $Msg(c) = \mathcal{M}$. Moreover, $A'(K_s)$ and $B'(z)$ represent the *continuations* of A and B, respectively, that will pre-

sumably communicate using the just established session key K_s. Notice that in $B'(z)$, variable z is bound to the received session key.

We want to analyze the execution of P with no intrusion; we thus consider $P \setminus \{c\}$, since the restriction guarantees that c is now a secure channel between A and B (no external attacker can access channel c). We obtain a process whose only possible execution is the correct one where A sends to B message $\{K_s, A\}_K$:

$$\begin{aligned} P \setminus \{c\} &\xrightarrow{\tau} (A'(K_s) \parallel [\langle \{K_s, A\}_K, K\rangle \vdash_{dec} w][w \vdash_{fst} z]B'(z)) \setminus \{c\} \\ &\equiv (A'(K_s) \parallel B'(K_s)) \setminus \{c\} \end{aligned}$$

Notice that, after the message exchange, the two continuations $A'(K_s)$ and $B'(K_s)$ share the correct session key K_s.

3 Non-interference for CryptoSPA

In this section we want to rephrase the non-interference theory developed for SPA to the richer setting of CryptoSPA. 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 study behavioural semantics for CryptoSPA over which the *GNDC* theory is parametrized.

3.1 Decorating Protocol Specifications

The first problem is to understand what are the high level actions and the low level ones in this setting. *NDC* 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, as we will see in the next section.

This idea of decorating a protocol with extra low level actions is formalized by considering a function γ, such that $\gamma(P)$ represents protocol P decorated with suitable low level actions. The decoration function γ should not affect the protocol behaviour, as its only purpose is to allow observing such a behaviour. This is achieved by explicitly requiring that $\gamma(P)$ behaves as P once the low level extra actions have been hidden (i.e., transformed into internal non-observable actions).

Let $f_{((I \cup O) \setminus C)}$ be the relabelling function that maps all the actions in $((I \cup O) \setminus C)$ into τ and let $\approx$ denote a suitable behavioural equivalence such that $P \approx P'$ only if P and P' behaves the same (equivalences of this kind will be formally defined in section 3.3).

Definition 1. *A function* $\gamma : \mathcal{E}_C \to \mathcal{E}$ *is a* decoration function *if for every CryptoSPA protocol* P *such that* $sort(P) \subseteq C$*, we have* $P \approx \gamma(P)[f_{((I \cup O) \setminus C)}]$.

We will see that the requirement above for γ functions may be easily achieved by considering γ's that decorate the protocols by only adding output actions, i.e., actions that simply make public some internal protocol values and do not modify the protocol execution by reading new values.

Example 2. Consider again the CryptoSPA protocol specification of example 1:

$$P = \overline{c}\,\{K_s, A\}_K.A'(K_s) \parallel c(y).[\langle y, K\rangle \vdash_{dec} w][w \vdash_{fst} z].B'(z)$$

Recall that it models the (flawed) key exchange protocol we presented in section 1.3:

$$A \to B : \{K_s, A\}_K$$

In such a section we discussed how both message authenticity and entity authentication might be verified by observing the suitable events $received(K_s)$ and $run(A, B), commit(B, A)$, respectively. Here we show how such events may be added to the specification P. We consider two decoration functions γ_{MA} and γ_{EA} such that:

$$\begin{aligned}
\gamma_{MA}(P) &= \overline{c}\,\{K_s, A\}_K.A'(K_s) \parallel c(y).[\langle y, K\rangle \vdash_{dec} w][w \vdash_{fst} z].\overline{received}(z).B'(z) \\
\gamma_{EA}(P) &= \overline{run}(A, B).\overline{c}\,\{K_s, A\}_K.A'(K_s) \\
&\quad \parallel c(y).[\langle y, K\rangle \vdash_{dec} w][w \vdash_{fst} z].\overline{commit}(B, A).B'(z)
\end{aligned}$$

$\gamma_{MA}(P)$ behaves as P apart from the fact that it sends the session key received by B on the low level channel *received*; $\gamma_{EA}(P)$ only modifies P by performing the two low level outputs $\overline{run}(A, B)$ and $\overline{commit}(B, A)$ when A starts the protocol and when B concludes it, respectively. Notice that these decorations do not change the protocol behaviour observed on channel c. As an example, $\gamma_{MA}(P)[f_{((I \cup O) \setminus C)}]$ corresponds to the protocol

$$\overline{c}\,\{K_s, A\}_K.A'(K_s) \parallel c(y).[\langle y, K\rangle \vdash_{dec} w][w \vdash_{fst} z].\tau.B'(z)$$

which is equivalent to P for any reasonable weak (i.e., ignoring τ's) behavioural equivalence.

3.2 The Enemy

Intuitively, an enemy 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. In principle, such a process could be modeled as a generic process X which can communicate only through the channels

belonging to C. However, in this way we obtain that X is a too powerful attacker. 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 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 information, as illustrated in the following example.

Example 3. Consider again the protocol P of Examples 1 and 2. Since only A and B know k_{AB}, this protocol should guarantee the authenticity of m_A even in the presence of an enemy, when only one protocol session is considered (we have seen that authenticity is not guaranteed for multiple sessions). We assume that $c \in C$ is a public channel and we consider the following attacker that belongs to $\mathcal{E}_C$:

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

Notice that this process may only communicate over the public channel c, i.e., $sort(X(m,a,k)) = \{c\}$. Consider now $X(K_X, A, k_{AB})$, which knows k_{AB} and can consequently send a faked message $\{K_X, A\}_{k_{AB}}$ to B. In order to observe this, we consider the decorated protocol $\gamma_{MA}(P)$ "under the attack" of X (note that we put X inside the scope of restriction):

$$(\gamma_{MA}(P) \,\|\, X(K_X, A, k_{AB})) \setminus C$$

After one τ communication step, the process above can perform $\overline{received}\,(K_X, A)$ which represents the fact that B has received K_X instead of m_A. This happens since we are considering an attacker $X(K_X, A, k_{AB})$ which is in some sense "guessing" k_{AB} (X knows k_{AB} in advance). As we are interested in attacks that can be carried out even when cryptography is completely reliable, we would like to forbid such infeasible attacks by restricting the set of admissible enemies. ■

The problem of guessing secret values can be solved by imposing some constraints on the initial data known by the intruders. Given a process P, we call $ID(P)$ the set of messages that occur syntactically in P. More formally, we define $ID(P)$ as $I(P, \emptyset)$, where $I : \mathcal{E} \times \mathcal{P}(Const) \longrightarrow \mathcal{P}(M)$ is given in Figure 4. Informally, $I(P,V)$ is a function that recursively visits the sub-terms of P and the body of the constants used. The argument V is used to check that the unwinding of a constant definition is performed only once.

Example 4. Consider $A(m_1)$, where $A(x) \stackrel{\text{def}}{=} \overline{c}\,x.\underline{0} \,\|\, \overline{c}\,m_2.A(m_3)$. Note that:

$$\begin{array}{lll}
I(A(m_3), \{A\}) & = \{m_3\} & \\
I(\overline{c}\,x.\underline{0}, \{A\}) & = I(\underline{0}, \{A\}) & = \emptyset \\
I(\overline{c}\,m_2.A(m_3), \{A\}) & = \{m_2\} \cup I(A(m_3), \{A\}) & = \{m_2, m_3\} \\
I(A(x), \{A\}) & = I(\overline{c}\,x.\underline{0}, \{A\}) \cup I(\overline{c}\,m_2.A(m_3), \{A\}) & = \{m_2, m_3\} \\
I(A(m_1), \emptyset) & = \{m_1\} \cup I(A(x), \{A\}) & = \{m_1, m_2, m_3\}
\end{array}$$

Thus, we have $ID(A(m_1)) = I(A(m_1), \emptyset) = \{m_1, m_2, m_3\}$. ■

$$
\begin{aligned}
I(\underline{0}, V) &= \emptyset \\
I(c(x).P, V) &= I(P, V) \\
I(\bar{c}\, e.P, V) &= \mathit{get\text{–}msg}(e) \cup I(P, V) \\
I(\tau.P, V) &= I(P, V) \\
I(P_1 + P_2, V) &= I(P_1, V) \cup I(P_2, V) \\
I(P_1 \parallel P_2, V) &= I(P_1, V) \cup I(P_2, V) \\
I(P \setminus L, V) &= I(P, V) \\
I(P[f], V) &= I(P, V) \\
I(A(e_1, \ldots, e_n), V) &= \begin{cases} \bigcup_{i \in \{1,\ldots,n\}} \mathit{get\text{–}msg}(e_i) \text{ if } A \in V \\ I(P, V \cup \{A\}) \cup \bigcup_{i \in \{1,\ldots,n\}} \mathit{get\text{–}msg}(e_i) \\ \qquad\qquad \mathit{otherwise} \end{cases} \\
&\quad \text{where } A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P \\
I([e = e']P_1; P_2, V) &= \mathit{get\text{–}msg}(e) \cup \mathit{get\text{–}msg}(e') \cup I(P_1, V) \cup I(P_2, V) \\
I([\langle e_1 \ldots e_r \rangle \vdash_{rule} x]P_1; P_2, V) &= (\textstyle\bigcup_{i \in \{1,\ldots,r\}} \mathit{get\text{–}msg}(e_i)) \cup I(P_1, V) \cup I(P_2, V)
\end{aligned}
$$

where

$$
\mathit{get\text{–}msg}(e) = \begin{cases} \{e\} & \text{if } e \text{ is a message} \\ \emptyset & \text{if } e \text{ is a variable} \end{cases}
$$

Fig. 4. Definition of $I(P, V)$.

Now, let $\phi_I \subseteq \mathcal{M}$ be the *finite*, initial knowledge that we would like to give to the intruders, 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.

Definition 2. *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 $\mathcal{E}_C^{\phi_I} = \{X \in \mathcal{E} \mid sort(X) \subseteq C$ and $ID(X) \subseteq \mathcal{D}(\phi_I)\}$.*

To see how $\mathcal{E}_C^{\phi_I}$ prevents the problem presented in Example 3, consider again the enemy $X(K_X, A, k_{AB})$ of that example. To indicate that k_{AB} is secret, we can now require that $k_{AB} \notin \mathcal{D}(\phi_I)$. Since $ID(X(K_X, A, k_{AB})) = \{K_X, A, k_{AB}\}$, we finally have that $X(K_X, A, k_{AB}) \notin \mathcal{E}_C^{\phi_I}$.

3.3 Behavioural Semantics

In this section we define some well-known behavioural semantics that we will use in order to formalize security properties. As mentioned in the Introduction, *NDC* as well as our general *GNDC* scheme is parametric with respect to the chosen semantic preorder (or equivalence). The relevant ones for this paper are essentially two: trace and weak bisimulation [38].

Trace Semantics. Most of the security properties that have been proposed for the analysis of security protocols are based on the simple notion of *trace*:

two processes are equivalent if they show exactly the same execution sequences (called *traces*). Let us consider the following relations between CryptoSPA terms: $P \stackrel{\tau}{\Longrightarrow} P'$ (or $P \Longrightarrow P'$) if $P \stackrel{\tau}{\longrightarrow}^* P'$, where $\stackrel{\tau}{\longrightarrow}^*$ is the reflexive (hence $P \Longrightarrow P$ always holds) and transitive closure of the $\stackrel{\tau}{\longrightarrow}$ relation; and then $P \stackrel{a}{\Longrightarrow} P'$ if $P \stackrel{\tau}{\Longrightarrow} \stackrel{a}{\longrightarrow} \stackrel{\tau}{\Longrightarrow} P'$.

For a trace $\sigma = a_1 \dots a_n$ we write $P \stackrel{\sigma}{\Longrightarrow} P'$ if $P \stackrel{a_1}{\Longrightarrow} P_1 \stackrel{a_2}{\Longrightarrow} \cdots \stackrel{a_{n-1}}{\Longrightarrow} P_{n-1} \stackrel{a_n}{\Longrightarrow} P'$ for some $P_1, \dots, P_{n-1}$. We say that P' is a *derivative* of P if there exists a trace σ such that $P \stackrel{\sigma}{\Longrightarrow} P'$.

The set $Tr(P)$ of *traces associated with* P is then defined as $Tr(P) = \{\sigma \in (Act \setminus \{\tau\})^* \mid \exists P' : P \stackrel{\sigma}{\Longrightarrow} P'\}$.

Definition 3. *Let $P, Q \in \mathcal{E}$. We write $P \leq_{trace} Q$ iff $Tr(P) \subseteq Tr(Q)$. We also say that P and Q are* trace equivalent *(notation $P \approx_{trace} Q$) iff $P \leq_{trace} Q$ and $Q \leq_{trace} P$.* ■

Example 5. Consider again the protocol P and the (too powerful) enemy process $X(K_X, A, k_{AB})$, both discussed in Example 3. It is easy to see that

$$(\gamma_{MA}(P) \parallel X(K_X, A, k_{AB})) \setminus C \not\approx_{trace} \gamma_{MA}(P) \setminus C$$

Indeed,

$$Tr((\gamma_{MA}(P) \parallel X(K_X, A, k_{AB})) \setminus C) = \{\overline{received}\,(K_X, A), \overline{received}\,(K_s, A)\}$$
$$Tr(\gamma_{MA}(P) \setminus C) = \{\overline{received}\,(K_s, A)\}$$

This proves that X is able to attack the protocol, by guessing the key k_{AB}. ■

We can also prove that the trace preorder is preserved by the parallel composition and by the restriction operator.

Proposition 1. *Let $P, Q, R \in \mathcal{E}$ be three processes and L a set of input channels. If $P \leq_{trace} Q$ then: $P \parallel R \leq_{trace} Q \parallel R$ and $P \setminus L \leq_{trace} Q \setminus L$.* ■

Weak Bisimulation. The general notion of *bisimulation* [38] consists of a mutual step-by-step simulation, i.e., given two processes E and F, when E executes a certain action moving to E' then F must be able to simulate this single step by executing the same action and moving to a term F' which is again bisimilar to E', and vice-versa. A weak bisimulation is a bisimulation which does not care about internal τ actions, i.e., when F simulates an action of E, it can also execute some τ actions before or after that action.

We write $E \stackrel{\hat{a}}{\Longrightarrow} E'$ for $E \stackrel{a}{\Longrightarrow} E'$ if $a \in \mathcal{L}$, and for $E(\stackrel{\tau}{\rightarrow})^* E'$ if $a = \tau$ (note that $\stackrel{\tau}{\Longrightarrow}$ requires at least one τ labelled transition while $\stackrel{\hat{\tau}}{\Longrightarrow}$ corresponds to $(\stackrel{\tau}{\rightarrow})^*$ and means zero or more τ labelled transitions).

The notion of *weak bisimulation* is defined as follows.

Definition 4 (Weak Bisimulation). *A binary relation $\mathcal{S} \subseteq \mathcal{E} \times \mathcal{E}$ over processes is a* weak bisimulation *if $(E, F) \in \mathcal{S}$ implies, for all $a \in Act$,*

- *whenever $E \xrightarrow{a} E'$, then there exists F' such that $F \stackrel{\hat{a}}{\Longrightarrow} F'$ and $(E', F') \in \mathcal{S}$;*
- *whenever $F \xrightarrow{a} F'$, then there exists E' such that $E \stackrel{\hat{a}}{\Longrightarrow} E'$ and $(E', F') \in \mathcal{S}$.*

Two processes $E, F \in \mathcal{E}$ are (weakly) bisimulation equivalent*, denoted by $E \approx_{bis} F$, if there exists a weak bisimulation $\mathcal{S}$ containing the pair (E, F).*

In [38] it is proved that $\approx_{bis}$ is the largest weak bisimulation and it is an equivalence relation.

NDC for CryptoSPA. The family of security properties $NDC_C^{\Phi_I}$ for CryptoSPA, for any choice of the initial knowledge Φ_I and for any choice of the behavioural semantics $\approx$ is defined as follows. To simplify the notation, we will write $P \|_C X$ as a shorthand for $(P \| X) \backslash C$.

Definition 5. *Let $P \in \mathcal{E}$ be a decorated CryptoSPA process and $\approx$ one of the behavioural equivalences presented in the previous section. We say that*

$$P \in NDC_C^{\Phi_I} \text{ iff } \forall X \in \mathcal{E}_C^{\phi_I} : P \underset{C}{\|} X \approx P \setminus C$$

On the one hand, $P \setminus C$ represents the secure specification of the protocol P running in isolation on perfectly secure channels. The visible behaviour of P is given by the property dependent, extra observable actions included into the specification. Hence, the behaviour of $P \setminus C$ should describe the security property of interest. On the other hand, if $P \setminus C$ is equivalent to $P \|_C X$, then this clearly means that X is not able to modify in any way the observable execution of P, i.e., the security property holds.

Strictly speaking, *NDC* is used to denote the specific instance in the family when $\approx$ is $\approx_{trace}$, and *BNDC* when $\approx$ is $\approx_{bis}$. Moreover, for notational convenience, if the behavioural semantics is a preorder $\lhd$, then the associated NDC variant is denoted with $NDC_{\lhd}$.

4 Verification and Compositionality

In this section we want to address the issue of formal verification of the *NDC* properties on cryptographic protocols.

First we will address the issue of the existence, in dependence of the behavioural semantics, of a *most powerful enemy* (*mpe*, for short), in such a way that the universal quantification over all possible enemies can be removed in favour of a single check against such a *mpe*.

We will also study the related problem of how to circumvent the universal quantification in the absence of a *mpe* (this is the case when the semantics is weak bisimulation) by, e.g., resorting to properties stronger than *NDC* (based on bisimulation).

Then we will address the issue of composition of secure cryptographic protocols: given two subprotocols that are *NDC* secure, under which conditions can we compose them to get a *NDC* secure protocol?

4.1 Most Powerful Enemy

A serious obstacle to the widespread use of *NDC* 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 seems 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 (with respect to the chosen behavioural semantics) than all the others, so that one can reduce the infinity of checks to just one, albeit huge, check with respect to such a most powerful enemy.

We will say that a preorder $\lhd$ is a *pre-congruence* (w.r.t. the operator $\|_C$) if for every $P, Q, R \in \mathcal{E}$ if $Q \lhd R$ then $P \|_C Q \lhd P \|_C R$. Thus it is easy to prove the following [37].

Proposition 2. *If $\lhd$ is a pre-congruence 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:*

$$P \in NDC_{\lhd} \qquad \textit{iff} \qquad P \underset{C}{\|} Top \ \lhd \ P\backslash C \qquad \blacksquare$$

Proof. ($\Leftarrow$) *By the hypothesis that $\lhd$ is a pre-congruence, we have that $X \lhd X'$ implies $P \|_C X \lhd P \|_C X'$. Thus, if for every process $X \in \mathcal{E}_C^{\phi_I}$ we have $X \lhd Top$, then we will also have $P \|_C X \lhd P \|_C Top$ for every process $X \in \mathcal{E}_C^{\phi_I}$, and so, as by hypothesis $P \|_C Top \lhd P\backslash C$, we obtain $P \|_C X \lhd P \|_C Top \lhd P\backslash C$, i.e., P is $NDC_{\lhd}$.*
($\Rightarrow$) *By definition of $NDC_{\lhd}$, since $Top \in \mathcal{E}_C^{\phi_I}$.* $\blacksquare$

If the hypotheses of the proposition above hold, then it is sufficient to check that $P\backslash C$ is equivalent to P composed to the most powerful enemy Top.

We also have the following corollary for the congruence induced by $\lhd$, which essentially clarifies that *NDC* is requiring that P under the attack of the most powerful enemy is equivalent to P under the attack of the less powerful enemy, as well as to P in isolation.

Corollary 1. *Let $\lhd$ be a pre-congruence and 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*

$$P \in NDC_{\approx} \qquad \textit{iff} \qquad P \underset{C}{\|} Bot \approx P \underset{C}{\|} Top \approx P\backslash C \qquad \blacksquare$$

Given these very general results, one may wonder if it is possible to apply them to some of the semantics we have described in Section 3.3. Indeed, this is the case, at least for the trace preorder $\leq_{trace}$, which is a pre-congruence (*cf* Proposition 1).

The easy part is to identify the minimal element *Bot* in $\mathcal{E}_C^{\phi_I}$ w.r.t. $\leq_{trace}$: it is clear that the minimum set of traces is the empty-set, that is generated, e.g., by process $\underline{0}$. As a matter of fact, $P \|_C \underline{0} \approx P\backslash C$, for most equivalences $\approx$.

Let us now try to identify the top element Top in $\mathcal{E}_C^{\phi_I}$ w.r.t. $\leq_{trace}$. A "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 $\mathcal{D}$. Note also that τ summands are not considered, as inessential for the trace preorder. The following holds.

Proposition 3. *If $X \in \mathcal{E}_C^{\phi}$ then $X \leq_{trace} Top_{trace}^{C,\phi}$.*

Proof. We prove that $\mathcal{R} = \{(X', Top_{trace}^{C,\phi'}) \mid X' \text{ is a derivative of } X, X' \in \mathcal{E}_C^{\phi'}\}$ is a (weak) simulation relation [38] containing the pair $(X, Top_{trace}^{C,\phi})$. As the simulation preorder is finer than the trace preorder, the thesis follows. There are three possible cases.

- *$X' \xrightarrow{c(m)} X''$ with $c \in C$, $m \in Msg(c)$, $X'' \in \mathcal{E}_C^{\phi''}$ and $\phi'' = \phi' \cup \{m\}$. Then, $Top_{trace}^{C,\phi'} \xrightarrow{c(m)} Top_{trace}^{C,\phi''}$ is derivable, and the pair $(X'', Top_{trace}^{C,\phi''}) \in \mathcal{R}$.*
- *$X' \xrightarrow{\overline{c}\, m} X''$ with $c \in C$, $m \in Msg(c) \cup \mathcal{D}(\phi)$ and so $X'' \in \mathcal{E}_C^{\phi'}$. Then, $Top_{trace}^{C,\phi'} \xrightarrow{\overline{c}\, m} Top_{trace}^{C,\phi'}$ is derivable, and the pair $(X'', Top_{trace}^{C,\phi'}) \in \mathcal{R}$.*
- *$X' \xrightarrow{\tau} X''$, hence $X'' \in \mathcal{E}_C^{\phi'}$. Then, $Top_{trace}^{C,\phi'} \stackrel{\hat{\tau}}{\Longrightarrow} Top_{trace}^{C,\phi'}$ is derivable, and the pair $(X'', Top_{trace}^{C,\phi'}) \in \mathcal{R}$.* ∎

So, we have proved that there exists a top of the set $\mathcal{E}_C^{\phi_I}$ with respect to $\leq_{trace}$ and it is indeed Top_{trace}^{C,ϕ_I}. Now we want to prove that the single check against the top element is enough to ensure *NDC*.

Corollary 2. *Let P be a CryptoSPA protocol. We have that $P \in NDC_C^{\Phi_I}$ iff $P \parallel_C Top_{trace}^{C,\phi_I} \approx_{trace} P \setminus C$.*

Proof. By Corollary 1, it is enough to observe that $\approx_{trace}$ is a precongruence (cf Proposition 1) and that the top element is Top_{trace}^{C,ϕ_I}. ∎

It is interesting to observe that Top_{trace}^{C,ϕ_I} is the top element also for other behavioural preorders, namely the simulation preorder [38] and the barbed precongruence [40]. Hence the theory above can be rephrased also for these observational semantics.

On the other hand, for weak bisimulation semantics it seems that no (manageable) *mpe* exist. Hence, in order to avoid the universal quantification we have to follow an alternative strategy, that is the subject of the next subsection.

Finally, observe that even if we have removed the universal quantification, we are not yet in the condition of easy verification of protocols because the *mpe* is in general an infinite state system. In order to perform an efficient verification, we need to adopt the strategy illustrated in the next subsection, using however a symbolic semantics (this is outside the scope of the current presentation).

4.2 Static Characterization of NDC

In this section we develop for *NDC* an alternative strategy to avoid the universal quantification over all the admissible enemies. The idea is based on a *static characterization* for *NDC* in the setting of SPA [17,19], according to which *NDC* is proved to coincide with another non interference property called *Strong Nondeterministic Non-Interference* (*SNNI* for short).

A SPA process P is $SNNI$ if $P\backslash H$, where no high level activity is allowed, behaves like system P where all the high level activities are *hidden* (i.e., transformed into internal τ actions). To express this second system, we need to introduce first the *hiding* operator P/L (where L is an arbitrary subset of $\mathcal{L}$), which is defined by means of the following rules.

$$\frac{P \xrightarrow{a} P'}{P/L \xrightarrow{a} P'/L}\ (a \notin L \cup \overline{L}) \qquad \frac{P \xrightarrow{a} P' \quad a \in L \cup \overline{L}}{P/L \xrightarrow{\tau} P'/L} \tag{1}$$

Now we are ready to define the property for SPA as follows: $P \in SNNI$ if and only if we have $P\backslash H \approx_{trace} P/H$. It is rather intuitive to see that P/H can be seen as $(P \parallel Top)\backslash H$, where Top is the top element of the trace preorder for SPA; hence, such a static characterization is somehow a corollary of the existence of a top element in the trace preorder (together with the fact that the trace preorder is a precongruence).

The main goal of this section is to show that such a result is translatable to CryptoSPA, with the proviso of handling the (initial) secrets properly. To this aim we first introduce a properly enhanced definition of the hiding operator $/^{\phi}$ for CryptoSPA that will allow us to define the non interference property $SNNI^{\phi}_{C}$, and then to prove that NDC^{ϕ}_{C} is the same as $SNNI^{\phi}_{C}$.

$$\frac{P \xrightarrow{a} P'}{P/^{\phi}C \xrightarrow{a} P'/^{\phi}C}\ (a \notin C \cup \overline{C}) \qquad \frac{P \xrightarrow{\overline{c}\,m} P' \quad c \in C \cup \overline{C} \quad m \in Msg(c) \quad \phi' = \phi \cup \{m\}}{P/^{\phi}C \xrightarrow{\tau} P'/^{\phi'}C} \qquad \frac{P \xrightarrow{c(m)} P' \quad c \in C \cup \overline{C} \quad m \in \mathcal{D}(\phi) \quad m \in Msg(c)}{P/^{\phi}C \xrightarrow{\tau} P'/^{\phi}C} \tag{2}$$

We are now ready to define the family of $SNNI^{\phi}_{C}$ properties, parametrized on the behavioural semantics $\approx$.

Definition 6. *A process P is $SNNI_C^\phi$ if $P\backslash C \approx P/^\phi C$.*

As for *NDC*, we will prefix $SNNI$ with a letter characterizing the actual equivalence $\approx$ we are considering; so, we will have $SNNI$ for trace, but $BSNNI$ for weak bisimulation. When the chosen behavioural semantics $\approx$ is actually $\approx_{trace}$, we have the following results.

Proposition 4. *If $\phi' \subseteq \phi$ then $SNNI_C^\phi \subseteq SNNI_C^{\phi'}$.*

Proof. Let P be a $SNNI_C^\phi$ process. In order to prove that $P \in SNNI_C^{\phi'}$, we have to check that $P\backslash C \approx_{trace} P/^{\phi'}C$, if $P\backslash C \approx_{trace} P/^\phi C$. The thesis follows by observing that $Tr(P\backslash C) \subseteq Tr(P/^{\phi'}C) \subseteq Tr(P/^\phi C)$. ■

Proposition 5. *$P \in NDC_C^\phi$ iff $P \in SNNI_C^\phi$.*

Proof. If $P \in NDC_C^\phi$, then by Corollary 2 we have $P \parallel_C Top_{trace}^{C,\phi} \approx_{trace} P\backslash C$. Hence, the thesis holds if $P \parallel_C Top_{trace}^{C,\phi} \approx_{trace} P/^\phi C$. We actually prove such an equality for the simulation equivalence $\asymp$ [38], an equivalence which is finer than trace equivalence. To this aim, we prove that the following two relations are (weak) simulations.

$$\mathcal{R}_1 = \{(P' \underset{C}{\parallel} Top_{trace}^{C,\phi'}, P'/^{\phi'}C) \mid P' \underset{C}{\parallel} Top_{trace}^{C,\phi'} \text{ is a derivative of } P \underset{C}{\parallel} Top_{trace}^{C,\phi}\}$$

and

$$\mathcal{R}_2 = \{(P'/^{\phi'}C, P' \underset{C}{\parallel} Top_{trace}^{C,\phi'}) \mid P'/^{\phi'}C \text{ is a derivative of } P/^\phi C\}$$

Let us start with $\mathcal{R}_1$. We have the following three cases.

- *$P' \parallel_C Top_{trace}^{C,\phi'} \xrightarrow{a} P'' \parallel_C Top_{trace}^{C,\phi'}$ with $a \notin C \cup \overline{C}$. Then, $P'/^{\phi'}C \xrightarrow{a} P''/^{\phi'}C$ and $(P'' \parallel_C Top_{trace}^{C,\phi'}, P''/^{\phi'}C) \in \mathcal{R}_1$.*
- *$P' \parallel_C Top_{trace}^{C,\phi'} \xrightarrow{\tau} P'' \parallel_C Top_{trace}^{C,\phi''}$ as $P' \xrightarrow{\overline{c}\,m} P''$ and $Top_{trace}^{C,\phi'} \xrightarrow{c(m)} Top_{trace}^{C,\phi''}$. Hence, $c \in C$, $m \in Msg(c)$, $\phi'' = \phi' \cup \{m\}$. Then, it is derivable $P'/^{\phi'}C \xrightarrow{\tau} P''/^{\phi''}C$ and $(P'' \parallel_C Top_{trace}^{C,\phi''}, P''/^{\phi''}C) \in \mathcal{R}_1$.*
- *$P' \parallel_C Top_{trace}^{C,\phi'} \xrightarrow{\tau} P'' \parallel_C Top_{trace}^{C,\phi'}$ as $P' \xrightarrow{c(m)} P''$ and $Top_{trace}^{C,\phi'} \xrightarrow{\overline{c}\,m} Top_{trace}^{C,\phi'}$. Hence, $c \in C$, $m \in Msg(c)$, and $m \in \mathcal{D}(\phi')$. Then, it is derivable $P'/^{\phi'}C \xrightarrow{\tau} P''/^{\phi'}C$ and $(P'' \parallel_C Top_{trace}^{C,\phi'}, P''/^{\phi'}C) \in \mathcal{R}_1$.*

Analogously for relation $\mathcal{R}_2$. ■

A consequence of the proposition above is that one may wonder why we should be interested to $SNNI_C^\Phi$ as, after all, it is equivalent to NDC_C^Φ. There are some good reasons of interest for $SNNI_C^\Phi$. First, even in the case of trace semantics, $SNNI_C^\Phi$ is of interest because it may lead to more efficient analysis, e.g., it is easier to

formulate in a symbolic way. Second, it helps the formulation of compositionality principle, as we will see in the following. Third, the new characterization does not consider explicitly the notion of (admissible) enemy, while it considers only the set of (initial) secrets. Hence, it is definable also for those preorders that do not admit the top element (most powerful enemy *mpe*).

This last comment deserves more explanation. Assume we want to prove that P satisfies $BNDC$; this may be useful, e.g., for security properties that need some form of liveness (e.g., non repudiation). Hence in order to avoid the universal quantification, we can try to prove that P satisfies the finer property of $SBSNNI_C^{\Phi}$, that requires that $BSNNI_C^{\Phi}$ holds for all the derivatives of P. In this way, as the state space of P is (usually) finite, we have removed the infinity of checks that $BNDC$ requires in favour of a finite number (one for each derivative of P) of $BSNNI_C^{\Phi}$ checks. A recent paper [7] has independently proposed this approach and shown how to efficiently prove $SBSNNI_C^{\Phi}$ by means of a new, equivalent property (called P_BNDC) that needs not to be verified on all the derivatives of P.

4.3 Compositionality

In this section we illustrate a compositional proof method for $SNNI_C^{\phi}$, originally introduced in [29]. Within the SPA theory, $SNNI$ is compositional, i.e. if $P, Q \in SNNI$ then $P \,\|\, Q \in SNNI$. Unfortunately, the same does not hold when considering enemies with limited knowledge, as for $SNNI_C^{\phi}$. For instance, consider the processes:

$$P = \overline{c_1} m_1.c_2(x)[x = m_2].\overline{c_3} m_2$$
$$Q = \overline{c_1} m_2.c_2(x)[x = m_1].\overline{c_3} m_1$$

Now, assuming $C = \{c_1, c_2\}$ and $\phi = \emptyset$, we have that $P, Q \in SNNI_C^{\phi}$; indeed, $P\backslash C$ is equivalent to $\underline{0}$, as well as $P/^{\phi}C$ (it can perform two τ's and then stop). However, $P \,\|\, Q \notin SNNI^{\phi}$. As a matter of fact, $P \,\|_C\, Q$ is equivalent to $\underline{0}$, while $(P \,\|\, Q)/^{\phi}C$ may perform both $\overline{c_3} m_1$ and $\overline{c_3} m_2$.

However, if we strengthen the assumptions we can get a compositional rule for establishing that a process belongs to $SNNI_C^{\phi}$. The *stability* assumption we make is that the process cannot increment its knowledge.

Definition 7. *We say that a process P is stable w.r.t. ϕ, whenever if $P/^{\phi}C \Longrightarrow P'/^{\phi'}C$ then $\mathcal{D}(\phi) = \mathcal{D}(\phi')$.*

Thus, the following proposition holds.

Proposition 6. *Assume that $P, Q \in SNNI_C^{\phi}$ and that P, Q are stable w.r.t. ϕ. Then $(P \,\|\, Q) \in SNNI_C^{\phi}$ and $P \,\|\, Q$ is stable w.r.t. ϕ.*

Example 6. We consider a simple example of the application of the principle above. Consider the processes $P = \overline{c}\{m\}_k.0$ and $Q = c?x.[x \quad k \vdash_{dec} z]\tau$. Consider also the process $KEN' = c?x.[x = m].\overline{public}\, m$. Assuming $\phi = \{\{m\}_k\}$

and $C = \{c\}$, we can establish that $P, Q, KEN' \in SNNI_C^{\phi}$ and are all stable; hence $P \,\|\, Q \,\|\, KEN' \in SNNI_C^{\phi}$, which means that $P \,\|\, Q$ keeps m secret. As a matter of fact, $(P \,\|\, Q \,\|\, KEN')\backslash C \approx_{trace} \underline{0}$.

Considering bisimulation semantics, it is not difficult to see that the compositionality principle above scales to $SBSNNI_C^{\phi}$.

Proposition 7. *Assume that $P, Q \in SBSNNI_C^{\phi}$ and that P, Q are stable w.r.t. ϕ. Then $P \,\|\, Q \in SBSNNI_C^{\phi}$ and $P \,\|\, Q$ is stable w.r.t. ϕ.*

5 A General Schema for Security Properties

In this section we formally define the $GNDC_{\lhd}^{\alpha}$ family of properties. The proposed family of security property is the following[4]:

Definition 8. *Let $P \in \mathcal{E}$ be a CryptoSPA process. We say that*

$$P \in GNDC_{\lhd}^{\gamma,\alpha} \text{ iff } \forall X \in \mathcal{E}_C^{\phi_I} : \gamma(P) \underset{C}{\|} X \lhd \alpha(P)$$

where $\lhd \in \mathcal{E} \times \mathcal{E}$ is a relation between processes, $\gamma : \mathcal{E}_C \to \mathcal{E}$ is a decoration function and $\alpha : \mathcal{E} \to \mathcal{E}$ is a function between processes. ■

We propose a sufficient criterion for a static characterization (i.e. not involving the universal quantifier $\forall$) of $GNDC_{\lhd}^{\gamma,\alpha}$ properties.

Proposition 8. *If $\lhd$ is a pre-congruence w.r.t. $\|_C$ 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:*

$$P \in GNDC_{\lhd}^{\gamma,\alpha} \qquad \textit{iff} \qquad \gamma(P) \underset{C}{\|} Top \lhd \alpha(P)$$ ■

In particular, if the hypotheses of the proposition above hold then it is sufficient to check that $\alpha(P)$ is satisfied when P is composed with the most general (i.e., most powerful) environment Top. The proposition above is a generalization to $GNDC$ of Proposition 3. The following corollary for the congruence induced by $\lhd$, generalizes Corollary 2:

Corollary 3. *Let $\lhd$ be a pre-congruence w.r.t. $\|_C$ and 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*

$$P \in GNDC_{\approx}^{\gamma,\alpha} \qquad \textit{iff} \qquad \gamma(P) \underset{C}{\|} Bot \approx \gamma(P) \underset{C}{\|} Top \approx \alpha(P)$$ ■

[4] Indeed $GNDC$ depends on the set $\mathcal{E}_C^{\phi_I}$, hence on ϕ_I and C, but we will omit them for the sake of readability.

5.1 Compositionality

We have a compositionality principle for the $GNDC^{\gamma,\alpha}_{\leq_{trace}}$ schema, also in this case under the assumption that the involved processes are stable. It basically states that security properties, represented as specification processes $\alpha_i(P_i)$ $i = 1, 2$, can be composed resulting in the property represented by the composition of the specification processes $\alpha_1(P_1) \parallel \alpha_2(P_2)$. As far as decorations are concerned, we assume that the compound process $P_1 \parallel P_2$ is decorated componentwise, hence below we omit the explicit mention to γs functions.

Proposition 9. *Given the set of initial knowledge ϕ and the set of public channels C, assume $P_i \in GNDC^{\alpha_i(P_i)}_{\leq_{trace}}$, with $i = 1, 2$, and P_1, P_2 are stable w.r.t. ϕ. It follows that $(P_1 \parallel P_2) \in GNDC^{\alpha_1(P_1) \parallel \alpha_2(P_2)}_{\leq_{trace}}$ and $P_1 \parallel P_2$ is stable w.r.t. ϕ.*

Example 7. Consider the following family of processes $S(i)$, each sending a message (m_i, i) after every time unit:

$$S(i) = [(m_i, i) \quad pk_A \vdash_{enc} z].\bar{c}z.S(i+1))$$

Consider also a family of receivers of this kind:

$$R(i) = c(y)[y \quad pk_A^{-1} \vdash_{dec} t][t \vdash_{snd} t_2][t_2 = i][t \vdash_{fst} t_1]\overline{out}\, t_1.R(i+1)$$

Basically, we have that $(S(0) \parallel R(0))\backslash C$, where $C = \{c\}$, is trace included into $Spec(0)$ where

$$Spec(i) = \overline{out}\, m_i.Spec(i+1)$$

Consider $\phi = \{\{(m_i, i)\}_{pk_A} \mid 0 \leq i\} \cup \{pk_A\}$ as the intruder's knowledge set. Then, we have that $S(0)$ and $R(0)$ are stable w.r.t. ϕ and $S(0) \in NDC^{\phi}_{C}$, with $S(0)\backslash C =_{trace} \underline{0}$, and $R(0) \in GNDC^{Spec(0)}_{\leq_{trace}}$. By Proposition 9, we have that

$$(S(0) \parallel R(0) \parallel Top^{C,\phi}_{trace})\backslash C \leq_{trace} \underline{0} \parallel Spec(0) \leq_{trace} Spec(0)$$

5.2 Non-deducibility on Compositions

Since $GNDC^{\gamma,\alpha}_{\lhd}$ is a generalization of *NDC* it can be instantiated in order to obtain *NDC* and also the *bisimulation* based *NDC*, called *BNDC*. Let Id_C be the identical decoration function. Then, *NDC* and *BNDC* correspond to $GNDC^{Id_C,E\backslash C}_{\approx_{trace}}$ and $GNDC^{Id_C,E\backslash C}_{\approx_{bisim}}$, respectively.

For *NDC* it is also possible to apply Corollary 3 obtaining an interesting static characterization.

Proposition 10. *P is NDC iff $P \parallel_C Top^{C,\phi_I}_{trace} \approx_{trace} P \setminus C$.* ∎

This result is the analogous of the one in [17], for multilevel security. Note that here we have found it as a particular case of the more general result of Corollary 3.

5.3 The Agreement Property

In this section we show that also the approach proposed in [33] for the analysis of authentication properties, inside the framework of CSP [31] process algebra, can be rephrased in terms of our specification schema. *Agreement* is a formalization, in the CSP calculus, of the *correspondence* idea proposed in [45], and previously discussed in section 1.3. The basic idea of the *Agreement* property is the following:

> "A protocol guarantees to a responder B *Agreement* with an initiator A on a set of data items ds if, whenever B (acting as responder) completes a run of the protocol, apparently with initiator A, then A has previously been running the protocol, apparently with B, and A was acting as initiator in her run, and the two agents agreed on the data values corresponding to all the variables in ds, and each such run of B corresponds to a unique run of A".

What is technically done in the *Agreement* property is to have for each party an action representing the running of the protocol and another one representing the completion of it. For example, consider an action $commit\_res(B, A, d)$ representing a correct termination of B as a responder that is convinced to communicate with A and agrees on data in d. Moreover we have an action $running\_ini(A, B, d)$ that represents the fact that A is running the protocol as initiator, apparently with B and with data d. If we have these two actions specified in the protocol, the *Agreement* property requires that when B executes $commit\_res(B, A, d)$ then A has previously executed $running\_ini(A, B, d)$. This means that every time B completes the protocol with A convinced that the relevant data are the ones represented by d, then A must have been running the protocol with B using exactly the data in d.

We can see the actions representing the runs and the commits as output actions over two particular channels *running_ini* and *commit_res*. In [33], it is assumed that the actions representing the runs and the commits are correctly specified in the protocol. Here, we consider protocols of the form:

$$P \stackrel{\text{def}}{=} C_1(U_1, U_1', d_1) \parallel \ldots \parallel C_n(U_n, U_n', d_n) \parallel V_1^{y_1}(Z_1, Z_1') \parallel \ldots \parallel V_n^{y_n}(Z_n, Z_n')$$

Where each *claimant* process $C_i(U_i, U_i', d_i)$ and each *verifier* process $V_i^{y_i}(Z_i, Z_i')$ are all sequential processes, i.e., processes composed only of sequences of actions, and represent user U_i willing to authenticate with user U_i', agreeing on data d_i, and user Z_i verifying the identity of user Z_i', agreeing on data contained in variable y_i. Given this specific (but reasonable) protocol form, we can easily give a decoration $\gamma_{Agree}(P)$ as follows:

$$\begin{aligned}\gamma_{Agree}(P) \stackrel{\text{def}}{=} & \overline{running\_ini}\,(U_1, U_1', d_1).C_1(U_1, U_1', d_1) \parallel \ldots \\ & \parallel \overline{running\_ini}\,(U_n, U_n', d_n).C_n(U_n, U_n', d_n) \\ & \parallel \tilde{V}_1(Z_1, Z_1', d_1') \parallel \ldots \parallel \tilde{V}_n(Z_n, Z_n', d_n')\end{aligned}$$

where $\tilde{V}_i$ is the process V_i in which $\underline{0}$ is replaced by $\overline{commit\_res}\,(Z_i, Z_i', y_i)$, i.e., it terminates with action $\overline{commit\_res}\,(Z_i, Z_i', y_i)$.

Now that we have defined how to decorate a protocol, we need to specify which are the good behaviours $\alpha(P)$, of a process P. For simplicity, we only analyze the case where A is the initiator and B is the responder, and the set ds of variables is composed only by d which can assume values in a set D. However, the specification can be easily extended in order to cover all the cases studied in [33]. Function $\alpha(P)$ can be defined as follows:

$$\begin{aligned} P' &= Top_{trace}^{Sort(P)\setminus\{running\_ini, commit\_res\},\mathcal{M}} \\ P''(x,y) &= \textstyle\sum_{d\in D} \overline{running\_ini}\,(x,y,d) \;\;.\;\; \overline{commit\_res}\,(y,x,d) \\ \alpha_{Agree}(P) &= P' \parallel P''(A,B) \end{aligned}$$

Given P, $\alpha(P)$ represents the most general system which satisfies the agreement property and has the same sort as P. As a matter of fact in $\alpha(P)$ the action $\overline{running\_ini}\,(A,B,d)$ always precedes $\overline{commit\_res}\,(B,A,d)$ for every datum d, and every combination of the other actions of P can be executed. In order to analyze more than one session, it is sufficient to consider an extended α which has several processes $P''(A,B)$ in parallel. For example, for n sessions we can consider the following:

$$\alpha_{Agree}(P) = P' \parallel \underbrace{P''(A,B) \parallel \ldots \parallel P''(A,B)}_{n}$$

We want that even in the presence of an hostile process X, P does not execute traces that are not in $\alpha(P)$,i.e., we require that $P \parallel_C X \leq_{trace} \alpha(P)$. So we can give the following definition:

Definition 9. *P satisfies Agreement iff P is $GNDC_{\leq_{trace}}^{\gamma_{Agree},\alpha_{Agree}}$.* ■

Example 8. Consider once more the protocol first presented in Example 1. It can be easily written in the form discussed above as follows:

$$\begin{aligned} P &= C(A,B,K_s) \parallel V^z(B,A) \\ C(a,b,k) &= \overline{c}\,\{k,a\}_K.\underline{0} \\ V^z(b,a) &= c(y).[\langle y,K\rangle \vdash_{dec} w][w \vdash_{fst} z].\underline{0} \end{aligned}$$

Notice that $V^z(B,A)$ specifies z as the variable which should contain the datum the processes are willing to agree on, i.e., the session key K_s. Now we have that

$$\begin{aligned} \gamma_{Agree}(P) = {} & \overline{running\_ini}\,(A,B,K_s).\overline{c}\,\{K_s,A\}_K.\underline{0} \\ & \parallel c(y).[\langle y,K\rangle \vdash_{dec} w][w \vdash_{fst} z].\overline{commit\_res}\,(B,A,z) \end{aligned}$$

It is possible to prove that $\forall X \in \mathcal{E}_C^{\phi_I} : \gamma_{Agree}(P) \parallel_C X \leq_{trace} \alpha_{Agree}(P)$, i.e., that $P \in GNDC_{\leq_{trace}}^{\gamma_{Agree},\alpha_{Agree}}$, if $K \notin \phi_I$. However, as noticed in Section 1.3, this protocol is flawed when more the one session is considered. Two sessions of P can be modelled by just replicating P twice, i.e., by considering $P \parallel P$.

Now let $X = c(w).\overline{c}\,(w).\overline{c}\,(w)$ be the enemy that intercepts the message sent over c and replays it twice. It is easy to see that $Tr(\gamma_{Agree}(P \,\|\, P) \,\|_C X) = \{\overline{running\_ini}\,(A, B, K_s).\overline{commit\_res}\,(B, A, K_s).\overline{commit\_res}\,(B, A, K_s)\}$ that is not included in $Tr(\alpha_{Agree}(P \,\|\, P))$, as the second commit is not matched by any running action. ■

Note that in [33] it is only required that *Agreement* holds when the system is composed with a particular intruder, which turns out to be equivalent to the most general one. In the following we exploit Proposition 8 in order to formally prove that such a (static) requirement is indeed sufficient (and necessary) to guarantee our *GNDC*-based version of *Agreement*. As a matter of fact, by Propositions 3 and 8 we immediately have the following result:

Proposition 11. *P satisfies Agreement iff it holds $\gamma_{Agree}(P) \,\|_C\, Top^{C,\phi_I}_{trace} \leq_{trace} \alpha_{Agree}(P)$.* ■

In [33], other versions of *Agreement* are defined. We can rephrase all of them in our model by simply changing the α function[5].

5.4 Message-Oriented Authentication

Now, we consider the message-based approach to authentication defined in [43,42] using the CSP language. The idea is to observe when a set of messages T authenticates another set of messages R. Informally, T authenticates R if the occurrence of some element of T implies the occurrence of some element of R (it is required that T and R are disjoint). When a system P satisfies this property we say that P satisfies T **authenticates** R.

In [43] the net is represented by a process $Medium$ which acts like a router by receiving and forwarding the messages to the correct process. In CSP, it is possible to observe the communication between the processes and the medium since they are not "internalized" as in CCS. However, we can simulate this by assuming that the $Medium$ echoes every routing of messages through particular output actions on two reserved channels *send* and *rcv* which do not belong to C. Action $\overline{send}\,(i, m)$ corresponds to the sending of message m performed by agent i and, symmetrically, $\overline{rcv}\,(i, m)$ represents the reception of it by agent i. (We assume to have a set *Agents* of agent identities with X denoting the intruder identity). In this way we can observe communication as done in CSP. This message echoing is obtained by suitably decorating the specification as follows:

$$\gamma_{auth}(P) = (P[f] \,\|\, Medium) \setminus W$$

$$Medium = \sum_{c \in C, i, j \in Agents} c^i_{send}(x).\overline{send}\,(i, x).(Medium \,\|\, \overline{c^j_{rcv}}\, x.\overline{rcv}\,(j, x).\underline{0})$$

[5] Indeed, *recentness* cannot be immediately rephrased in our CCS-based model, because of the difference in handling communication with respect to CSP. This could be overcome by extending our language with time as done in [21,29]. This is only related to the differences in the model, and is not caused by a weakness of our schema.

where f is a relabelling function that maps all the output actions $\overline{c}(m)$, performed by agent i, into $\overline{c^i_{send}}(m)$ and all the input actions $c(m)$, performed by agent i, into $c^i_{rcv}(m)$[6]. Moreover, $W = \{c^i_* \mid i \neq X \text{ and } * = send, rcv\}$, i.e., W is the set of all the $Medium$ channels that are not used to communicate with the intruder X. This relabelling has the effect of forcing all the communication through the Medium, which can consequently make observable every message exchange through the special channels *send* and *rcv*. Notice that messages are buffered so that the Medium is always ready to "route" new messages.

Sets T and R range over these reserved actions. We can now define the $\alpha_{auth^T_R}(P)$ function as follows:

$$\begin{aligned} \alpha_{auth^T_R}(P) &= P' \\ P' &= (\sum_{\substack{a \in Act \\ a \notin R \cup T}} a.P') + \sum_{\substack{a \in Act \\ a \in R}} a.P'' \\ P'' &= Top^{Sort(P),\mathcal{M}}_{trace} \end{aligned}$$

Process $\alpha_{auth^T_R}(P)$ can execute actions in T only after it has executed some actions in R. This can be seen by noticing that $\alpha_{auth^T_R}(P)$ moves to P'' (which can execute also actions in T) only after it performs at least one action in R. This is exactly what we require by our system P and $\alpha_{auth^T_R}(P)$ is indeed the most general system (with the same sort as P) satisfying T **authenticates** R. So we can give the following definition:

Definition 10. *P satisfies T* **authenticates** *R iff P is $GNDC^{\gamma_{auth},\alpha_{auth^T_R}}_{\leq_{trace}}$* ■

As in the section above, we can prove that the approach followed in [43], where it is considered only the most powerful intruder, guarantees that the property holds in the presence of whatever hostile process. By Propositions 3 and 8 we obtain that:

Proposition 12. *P satisfies T* **authenticates** *R iff $\gamma_{auth}(P) \parallel_C Top^{C,\phi_I}_{trace} \leq_{trace} \alpha_{auth^T_R}(P)$.* ■

5.5 Secrecy

In this section we show that *NDC* can be easily adapted for analysing secrecy in networks. Consider now 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

[6] We are implicitly assuming that channels with subscripts *send* and *rcv* are not used in protocol specification. This assumption is trivially met by a syntactical renaming of channels. We are also assuming that every agent has a unique name i. This is also done in [43] and will be exploited when considering non-repudiation.

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 decorate $P(m)$ as follows:

$$\gamma_{secret}(P(m)) = P(m) \parallel KN(m)$$

Intuitively, $\gamma_{secret}(P(m))$ is a modified protocol where the learning of m is now notified. To guarantee the secrecy of m, it is enough to require that for every secret M and for every enemy X, process $(\gamma_{secret}(P(M)) \parallel X) \setminus C$ never executes a $\overline{learnt}\, M$ action. On the one hand, if $(\gamma_{secret}(P(M)) \parallel X) \setminus C$ executes $\overline{learnt}\, M$ then M has been sent over the public channel c_k by either $P(M)$ or X. In both cases the message is not secret anymore. In the former situation $P(M)$ is making M public, while in the latter X has for sure learned M before sending it over c_k. On the other hand, if an enemy X is able to learn message M then there also exists an enemy X' that will send such a message over channel c_k and thus $(\gamma_{secret}(P(M)) \parallel X') \setminus C$ will eventually execute $\overline{learnt}\, M$.

This can be formalized by considering a simple function α_{secret} that can execute all the protocol actions but the special action *learnt*:

$$\alpha_{secret}(P) = Top_{trace}^{Sort(P),\mathcal{M}}$$

Definition 11. *$P(m)$ preserves the secrecy of m iff for all (secret) messages $M \in \mathcal{M} \setminus \mathcal{D}(\phi_I)$ $P(M)$ is $GNDC_{\leq_{trace}}^{\gamma_{secret},\alpha_{secret}}$.* ■

Notice that $\alpha_{secret}(P(M))$ never executes action *learnt*, thus forbidding its execution even when every possible intruder is considered.

An Example. In this section we show through a simple example how the *NDC*-based secrecy verification works. We consider a simplified version of the Wide Mouthed Frog Protocol [9].

Consider two processes A and B respectively sharing keys k_{AS} and k_{BS} with a trusted server S. In order to establish a secure channel with B, A sends a fresh key k_{AB} encrypted with k_{AS} to the server S. Then, the server decrypts the key and forwards it to B, this time encrypted with k_{BS}. Now B has the key k_{AB} and A can send a message m_A encrypted with k_{AB} to B. The protocol is composed of the following three messages (see also Figure 5):

$$\begin{array}{ll} \text{message 1} & A \rightarrow S : A, B, \{k_{AB}\}_{k_{AS}} \\ \text{message 2} & S \rightarrow B : \{A, k_{AB}\}_{k_{BS}} \\ \text{message 3} & A \rightarrow B : \{m_A\}_{k_{AB}} \end{array}$$

The main differences with respect to the original protocol is that here messages 1 and 2 do not contain timestamps (as studied in [2] for authentication). Moreover,

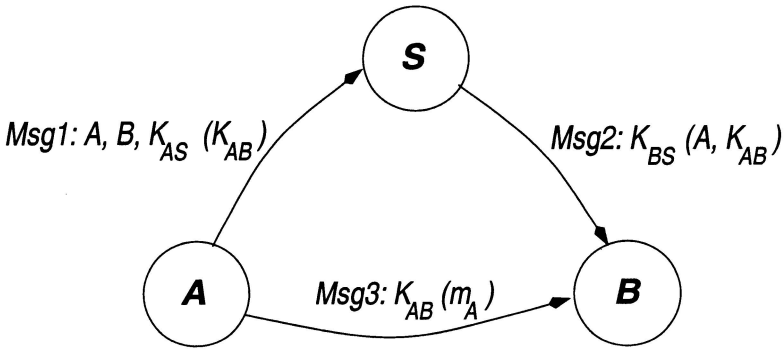


Fig. 5. Graphical description of the *WMF* protocol.

in message 1 the identifier B is sent as plaintext while in the original protocol it is encrypted with the session key (this modification generates, as we will show, a secrecy attack). We specify the protocol as follows[7]:

$$\begin{aligned} A(m,k) &\stackrel{\text{def}}{=} \overline{c_1}\,((A,B),\{k\}_{k_{AS}})\;.\;\overline{c_3}\,\{m\}_k \\ B &\stackrel{\text{def}}{=} c_2(y)\;.\;[\langle y,k_{BS}\rangle \vdash_{dec} z]\;[z \vdash_{snd} s]\;c_3(t)\;.\;[\langle t,s\rangle \vdash_{dec} w] \\ S &\stackrel{\text{def}}{=} c_1(x)\;.\;[x \vdash_{fst} i]\;[i \vdash_{fst} s]\;[i \vdash_{snd} r] \\ &\quad [x \vdash_{snd} c]\;[\langle c,K(s)\rangle \vdash_{dec} z]\;\overline{c_2}\,\{(s,z)\}_{K(r)}\;.\;S \\ P(n) &\stackrel{\text{def}}{=} A(n,k_{AB}) \parallel B \parallel S \end{aligned}$$

where $K(id)$ is a function that returns the key shared between the server and entity id (e.g., $K(A)$ returns k_{AS}). Moreover we have that $\{c_1,c_2,c_3\} \subseteq C$. Consider now the following enemy:

$$\begin{aligned} \tilde{X} \stackrel{\text{def}}{=}\; & c_1(x)\;[x \vdash_{snd} y] && \texttt{\% intercepts message 1} \\ & \overline{c_1}\,((A,E),y). && \texttt{\% replaces B with E, sends it} \\ & c_2(z)\;[\langle z,k_{ES}\rangle \vdash_{dec} w]\;[w \vdash_{snd} k]. && \texttt{\% intercepts msg 2, obtains k} \\ & c_3(j)[\langle j,k\rangle \vdash_{dec} m] && \texttt{\% decrypts msg 3} \\ & \overline{c_k}\,m && \texttt{\% sends message to KN} \end{aligned}$$

It is easy to see that process $(\gamma_{secret}(P(M)) \parallel \tilde{X}) \setminus C \not\leq_{trace} \alpha_{secret}(P(M))$ as the former process can execute $\overline{learnt}\,M$. The attack performed by $\tilde{X}$ is the following:

$$\begin{array}{lll} \text{message 1} & A \to E(S) & : A,B,\{K_{AB}\}_{K_{AS}} \\ \text{message 1}' & E(A) \to S & : A,E,\{K_{AB}\}_{K_{AS}} \\ \text{message 2}' & S \to E & : \{A,K_{AB}\}_{K_{ES}} \\ \text{message 3} & A \to E(B) & : \{M\}_{K_{AB}} \end{array}$$

[7] We encode tuples through a left associative canonical form, e.g., the first message $A,B,\{k_{AB}\}_{k_{AS}}$ becomes $((A,B),\{k_{AB}\}_{k_{AS}})$.

By message $2'$ the enemy learns K_{AB} and, by message 3 which is addressed to B, it finally learns M.

Consider now the protocol where, in the first message, B is encrypted with the session key:

$$\text{message 1} \qquad A \to S : A, \{B, K_{AB}\}_{K_{AS}}$$

Here the secrecy attack is not possible anymore since cryptography protects the identifier from being modified by the intruder.

5.6 Non-repudiation

In this section, we show that also non-repudiation properties can be formulated within the *GNDC* schema. Non repudiation protocols aim at producing evidence about the execution of services, among parties that do not trust each other (see [46,47]).

In [41] Schneider shows how to apply verification methods based on *CSP* process algebra to the analysis of a (fair) non repudiation protocol proposed in [46]. Among the non repudiation properties studied in [41,47], we briefly recall:

- *Non Repudiation of Origin (NRO)* is intended to protect the receiver from the false denial of another party to have sent a message.
- *Non Repudiation of Receipt (NRR)* is intended to protect the sender form the false denial of another party to have received a message.

Intuitively, the analysis performed by Schneider is similar to his message based authentication (see section above). As an example, consider NRO verification: if the receiver is able to produce an evidence of the sending of a certain message m, then m should have been sent. In other words, such an evidence should "authenticate" m (in the sense of message-based authentication).

Non-repudiation protocols differ from the protocols discussed so far, which always involve communication among two or more trusted parties in an hostile environment. In non-repudiation protocols, parties do not trust each other, and in particular, one of them could try to act maliciously in order to obtain some advantage. As we will see, this can be modelled in the *GNDC* schema by considering the malicious party as a whole with the hostile environment.

In the verification of NRO (NRR) we assume that a Judge should be able to establish that a certain message has been sent (received) if he obtains some evidence of it from the receiver (sender). This verification should be carried out by only assuming that both the sender and the receiver have not sent on the net some secret information which could invalidate the evidence, like, e.g., their signature keys. In case of a dispute, the Judge cannot assume that both the parties have followed the protocol but she will always assume that none of them has compromised his own secret key.

Schneider models both the sender and the receiver similarly to the most general intruder with the constraint that long-term keys are never compromised. In order to apply the *GNDC* schema, we consider a weaker (but still reasonable)

notion of NRO; in particular, we require that if the receiver B, after following the protocol, is able to give evidence of origin, then the sender A has actually sent that message. We call this NRO with honest receiver written NRO_{hr}. It can be encoded in the $GNDC$ schema by considering a process P_B where only the receiver B and the fragment of A, T_A, related to encryption with long-term (secret) keys, are specified. P_B has the following form:

$$P_B \stackrel{\text{def}}{=} T_A \parallel \ldots \parallel T_A \parallel B_1^{y_1}(Z_1, Z_1') \parallel \ldots \parallel B_n^{y_n}(Z_n, Z_n')$$

Where each process $B_i^{y_i}(Z_i, Z_i')$ is sequential and represent Z_i getting the evidence of origin of the message contained in variable y_i sent by user Z_i'. Given this specific (but reasonable) protocol form, we can easily give a decoration $\gamma_{nro}(P_B)$ as follows:

$$\gamma_{nro}(P_B) \stackrel{\text{def}}{=} \gamma_{auth}(\tilde{P}_B)$$

where γ_{auth} is the decoration function defined in section 5.4 for message-oriented authentication, and $\tilde{P}_B$ is process P_B in which all the $\underline{0}$'s of processes $B_i^{y_i}(Z_i, Z_i')$ are replaced by $\overline{ev\_of\_or}(Z_i, Z_i', y_i)$. For instance, action $\overline{ev\_of\_or}(B, A, m)$ is the action which signals that B has evidence of origin of m from A. This latter decoration is similar to the one we used for adding commit actions in Section 5.3.

Now, recall that in process $Medium\ send(A, m)$ represents the sending of message m by agent A. Then α_{nro} can be simply defined as $\alpha_{auth}^{\{ev\_of\_or(B,A,m)\}}_{\{send(A,m)\}}$.

Definition 12. *P_B guarantees NRO_{hr} iff P_B is $GNDC^{\gamma_{nro},\alpha_{nro}}_{\leq_{trace}}$*

We show how this definition works through a simple example.

Example 9. A typical way of guaranteeing non-repudiation of origin, is to use digital signature. Consider the message exchange:

$$\text{message 1} \qquad A \rightarrow B : M, sign_A(M)$$

Since only A may generate $sign_A(M)$, then B is guaranteed that A has originated such a message. To see how the formalization of non-repudiation works, we consider a flawed version of the protocol above, in which shared key is used instead of signature:

$$\text{message 1} \qquad A \rightarrow B : M, MAC_K(M)$$

K is a key shared between A and B. The protocol uses a *Message Authentication Code* (MAC), i.e., a keyed hash function such that $MAC_K(M)$ can be efficiently calculated if and only if the key K is known. A MAC can be modelled in SPA by making the inverse key $k^{-1} \neq k$ not available to both the principals and the enemy. This is a simple way of simulating the unidirectionality provided by the (one-way) hashing. This protocol does not guarantee non-repudiation since both A and B might have generated $MAC_K(M)$. We specify B as follows:

$$Responder^j(b, a, k) \stackrel{\text{def}}{=} c(x).[x \vdash_{fst} i]\ [x \vdash_{snd} j]\ [\{i\}_k = j].\underline{0}$$

As mentioned above, we also need to model the honest fragment of A, i.e., the part of A that deals with long-term keys. Indeed, A can never repudiate a message by declaring she has erroneously disclosed her long-term key. It is her responsibility to keep keys securely stored. The remaining part of Alice is not modelled in order to let her (possibly) behave dishonestly:

$$Initiator(a,k) \stackrel{\text{def}}{=} d(x).\overline{d}\,(\{x\}_k).Initiator(a,k)$$

Notice that this honest fragment of A uses a new channel d. This channel is used to communicate with the intruder which implicitly describes the dishonest fragment of A.

We first consider $P_B = Initiator(A,K) \parallel Responder^j(B,A,K)$, in which A and B only play the Initiator and Responder roles, respectively. Since A is the only entity which generates messages encrypted with key k, NRO is guaranteed, i.e., every action $\overline{ev\_of\_or}\,(B,A,j)$ executed by $\gamma_{nro}(P_B)$ will be preceded by a $\overline{send}\,(A,j)$ generated by the $Medium$. Notice that $Responder^j(B,A,K)$, once decorated by γ_{nro}, executes a $\overline{ev\_of\_or}\,(B,A,j)$ only if j is of the form $\{i\}_K$, i.e., only if the MAC is valid.

To see a potential vulnerability of this protocol it is sufficient to consider two parallel sessions with exchanged roles:

$$\begin{aligned} P'_B = {} & Initiator(A,K) \parallel Responder^j(B,A,K) \parallel \\ & Initiator(B,K) \parallel Responder^j(A,B,K) \end{aligned}$$

Consider now the intruder $X_A = \overline{d^X_{send}}\,(M).d^X_{rcv}(x).\overline{c^X_{send}}\,(M,x).\underline{0}$, which asks one of the initiators to produce a MAC of M and then sends the pair $(M, MAC_K(M))$ to one of the responders. A non-repudiation problem arises when the same entity is producing the MAC and checking it. We show an execution sequence where B is convinced to receive a MAC from A but he is actually the creator of the MAC. Recall that $\gamma_{nro}(P'_B)$ introduces the medium $Medium$ of section 5.4 in between the communicating parties.

$$\begin{aligned}
& X_A \parallel \gamma_{nro}(P'_B) \\
& \xrightarrow{\overline{send}\,(X,M)\ \overline{rcv}\,(B,M)\ \overline{send}\,(B,\{M\}_K)} d^X_{rcv}(x).\overline{c^X_{send}}\,(M,x).\underline{0} \parallel \gamma_{nro}(P'_B) \\
& \qquad\qquad \parallel \overline{d^X_{rcv}}\,\{M\}_K.\overline{rcv}\,(X,\{M\}_K).\underline{0} \\
& \xrightarrow{\overline{rcv}\,(X,\{M\}_K)} \overline{c^X_{send}}\,(M,\{M\}_K).\underline{0} \parallel \gamma_{nro}(P'_B) \\
& \xrightarrow{\overline{send}\,(X,(M,\{M\}_K))} \gamma_{nro}(P'_B) \parallel \overline{c^B_{rcv}}\,(M,\{M\}_K).\overline{rcv}\,(B,(M,\{M\}_K)).\underline{0} \\
& \xrightarrow{\overline{rcv}\,(B,(M,\{M\}_K))\ \overline{ev_of_or}\,(B,A,\{M\}_K)} \gamma_{nro}(Initiator(A,K) \parallel Initiator(B,K) \\
& \qquad\qquad \parallel Responder^j(A,B,K))
\end{aligned}$$

The execution trace is:

$$\overline{send}(X,M), \overline{rcv}(B,M), \overline{send}(B,\{M\}_K), \overline{rcv}(X,\{M\}_K),$$
$$\overline{send}(X,(M,\{M\}_K)), \overline{rcv}(B,(M,\{M\}_K)), \overline{ev\_of\_or}(B,A,\{M\}_K)$$

corresponding to the enemy X sending M to B playing the initiator role, B responding with $\{M\}_K$ and, finally, X forwarding message $\{M\}_K$ to B playing the responder role, who finally executes $\overline{ev\_of\_or}(B,A,\{M\}_K)$.

Notice that there is no $\overline{send}(A,\{M\}_K)$ matching the $\overline{ev\_of\_or}(B,A,\{M\}_K)$ event (indeed A is doing nothing). This is due to the fact that the encrypted message $\{M\}_K$ has been generated by B himself (in the Initiator role) at the execution step corresponding to $\overline{send}(B,\{M\}_K)$. Thus this trace is not a trace of $\alpha_{nro}(P'_B)$, proving that P'_B does not guarantee NRO. ■

Analogously, we define non-repudiation of origin with honest sender, i.e., NRO_{hs}. This property can be encoded in the *GNDC* schema by simply considering process P_A instead of P_B. *NRO* can be defined as the intersection of NRO_{hr} and NRO_{hs}, i.e., $P_A, P_B \in GNDC^{\alpha_{nro}}_{\leq_{trace}}$.

An analogous definition may be given for *weak-NRR*, even though the situation is slightly more complicated since, in this case, also liveness properties should be considered. In fact, it is not necessary that the message has been effectively received: it is sufficient to require that the message is "available" for reception, through, e.g. , a Trusted Third Party which makes the message downloadable, as proposed in [46]. We do not address this issue in details, since we prefer to focus our attention to another property which is also based on liveness. The property is *fairness* [47]:

– *Fairness*: At no point in the protocol run does either of participants have an advantage. In other words no one of the party can get his own evidence and avoid the other to get his corresponding evidence.

As observed in [41], this property cannot be defined as a safety property (i.e. nothing bad happens). Indeed we have to prove that whenever one of the two participants obtains his own evidence, then the other must be in the position to get his own evidence too. This can be seen as a liveness property (i.e. something good happens). For this reason, in the analysis of this property it is used the *failure* model [31] instead of the trace one. As a matter of fact, failure equivalence is actually able to observe potential deadlocks in the executions, and so it permits to see if something can be executed or not (i.e., if an evidence can be obtained or not).

The verification technique for the fairness property proposed in [41] directly fits in the *GNDC* schema. Indeed, it is reasonable to assume that an agent can require fairness from the other one only in the case he behaves correctly, i.e., if it follows the protocol. For example, the *fairness* for the sender A of receiving evidence of message receipt can be defined in the *GNDC* schema by considering a suitable relation $\leq_{failure}$ which takes into account failures, and a function α_{fair} which models the fact that after the receiver gets evidence of the origin (ev_of_or) then the sender has the possibility to obtain his own

evidence of receipt (ev_of_re). This is modeled in α_{fair} by making the action ev_of_re always executable after that ev_of_or has been engaged. We show how this formalization of fairness works, through a simple example.

Example 10. Consider the following non-repudiation protocol, where Bob requires non-repudiation of origin and Alice requires non-repudiation of receipt of the same message M:

$$\begin{array}{ll} \text{message 1} & A \rightarrow B : B, M, sign_A(B, M) \\ \text{message 2} & B \rightarrow A : A, sign_B(A, M) \end{array}$$

In order for this protocol to be fair, we need to guarantee that whenever Alice gets her evidence of receipt, then Bob eventually gets his evidence of origin, and vice-versa. If A and B execute the ev_of_re and ev_of_or actions just after checking the received signatures, we easily find out that the protocol does not guarantee fairness: it is sufficient for B to quit the protocol session after he has received his evidence of origin. This corresponds to an execution trace in which ev_of_or is executed but no ev_of_re is possible after it. This failure in executing the latter action is revealed by the failure preorder, and, consequently, the attack is captured. We leave the formalization of this example as an exercise to the interested reader. ■

In [7,8], an extension of the bisimulation-based *BNDC* property is applied to the verification of non-repudiation protocols.

5.7 Authentication in the Spi-Calculus

In [2,1] an interesting notion of authentication is proposed. The basic idea is the following: consider a protocol $P(M)$, which tries to transmit message M from one party (say A) to another one (say B). The authentication of the message M is checked by verifying if $P(M)$ is equivalent to a specification $P_{spec}(M)$ where M is always delivered correctly. In $P_{spec}(M)$ the receiver B always knows M and whatever happens on the communication channel, B will continue its execution exactly as it had received the correct message M. In other words, $P_{spec}(M)$ represents the situation where M is always received and no enemy is able to replace it with a different message. If $P(M)$ is equivalent to $P_{spec}(M)$ then also $P(M)$ is clearly able to avoid any possible attack. The language used in [2,1] is the spi-calculus. Moreover the may-testing equivalence [11], denoted with $\approx_{may}$, is used in order to check that $P(M)$ is equivalent to $P_{spec}(M)$ with respect to any possible interaction with the (hostile) environment. The definition of authentication in the spi-calculus has the following form: $P(M)$ guarantees spi-authentication if and only if for all M we have that

$$P(M) \approx_{may} P_{spec}(M)$$

There are many similarities between spi-calculus authentication and *NDC*: both properties are based on a notion of behavioral equivalence; moreover, they both check whether the "process under attack" behaves like a secure specification. It is however important to notice that this is done in a quite different

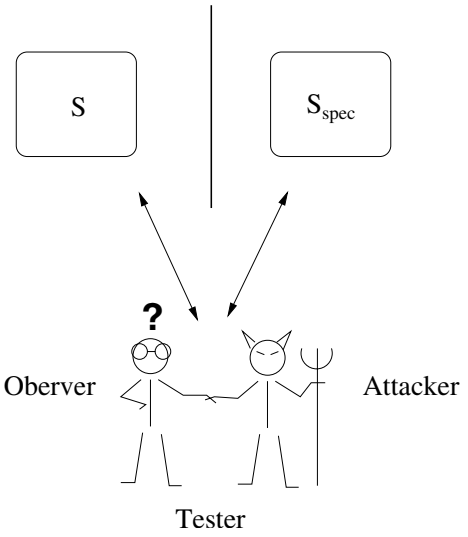


Fig. 6. Testers as "observers and attackers" in spi-authentication.

way. In the spi-calculus the process is implicitly checked against all the possible interactions with the (hostile) environment through the use of the may-testing equivalence. There, the tester plays simultaneously both the role of the attacker and the role of the observer (see Fig. 6). On the other hand, the *NDC*-based approach performs an explicit quantification over all possible intruders, then observing the outcome of the attack (see Fig. 7).

In [20] we have proved that spi-authentication can be equivalently expressed as an instance of *NDC*. To see how this is done it is useful to write a protocol in a particular style that we call *normal form*. We assume that, after delivering message M, protocol $P(M)$ executes a *continuation* process $F(M)$. In general, more than one continuation could be present. Given a protocol S, we denote all of its occurrences of continuations as $\{F_1(x_1), \ldots, F_n(x_n)\}$, where x_i represents the only free variable of F_i. From S we derive a process $S_{nf}(m_1, \ldots, m_n)$ in normal form as follows:

$$(S' \parallel \Pi_{i \in 1 \ldots n}\ p_{F_i}(x_i).F_i(x_i)) \setminus \boldsymbol{p} \tag{3}$$

where S' is the process S where every continuation $F_i(x_i)$ is replaced by $\overline{p_{F_i}}(x_i)$, and $\boldsymbol{p} = \{p_{F_1}, \ldots, p_{F_n}\}$ is a set of channels that are used neither in S nor in F_i and are not contained in C. Note that the channels in $\boldsymbol{p}$ are indexed with the continuations F_i. This is useful for managing multiple concurrent sessions between senders and receivers which can be modeled by considering n copies of the sender and n copies of the receiver in parallel. We assume that syntactically equal continuations (up to renaming of bound variables, i.e., α-conversion) correspond to the same protocol between two users but in different parallel sessions.

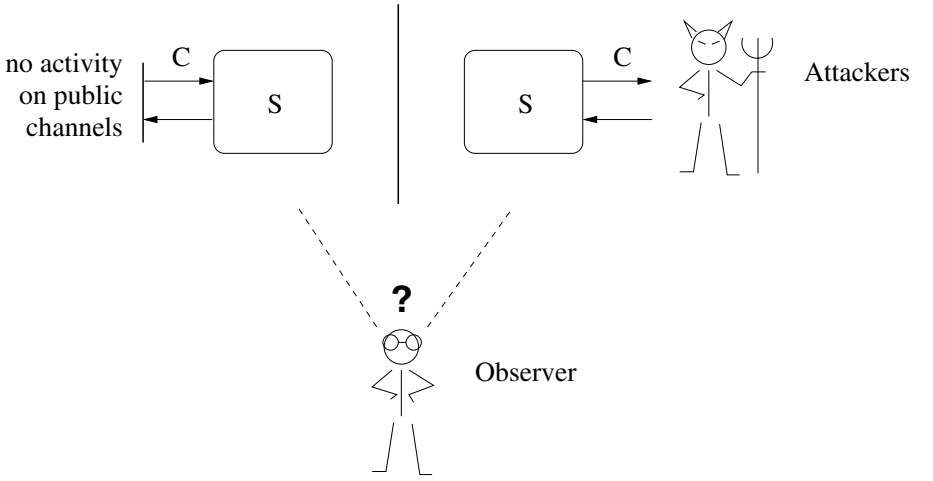


Fig. 7. Separated attackers and observers in *NDC*-authentication.

Given this particular form for protocols, it is quite natural to derive a secure specification. More precisely, given the normal form $S_{nf}(m_1, \ldots, m_n)$ as in (3), it is sufficient to define $S_{spec}(m_1, \ldots, m_n)$ as follows:

$$(S' \parallel \Pi_{i \in 1 \ldots n}\ p_{F_i}(x_i).[x_i = m_i]F_i(m_i)) \setminus \boldsymbol{p} \tag{4}$$

Note that every continuation is enabled only if the received message x_i is equal to the correct message m_i. Note also that in the case of multiple sessions, this simply requires that a "correct" multiset of messages is delivered from one process to another one, in whatever possible order (see the example below in this section).

We require some reasonable properties of the specification S_{spec}. First we require that it guarantees *NDC*, for all vector of messages $\mathbf{m}$. In [20] we prove that any specification S_{spec} guarantees *NDC*, under the following well-formedness condition:

WFC1 For every vector of messages $\mathbf{m}$, $S_{spec}(\mathbf{m}) \setminus C \approx_{may} \Pi_{k \in 1..n} F_k(\mathbf{m}_k)$

This condition is a very natural one as it requires that all the continuations of the specifications when there is no attacker at all, are eventually enabled. In other words, the specification is well-formed for not containing unreachable continuations. If it does, then some useless redundancy is present in S_{spec}.

We have an additional well-formedness condition:

WFC2 For every vector of messages $\mathbf{m}$, $S(\mathbf{m}) \setminus C \approx_{may} S_{spec}(\mathbf{m}) \setminus C$

The condition above is an obvious requisite for S_{spec}: the process S and its specification S_{spec} behave in the same way when the public channels are protected through the restriction (i.e., when no attack is possible).

Let $S^{\boldsymbol{F'}}$ denote the protocol S where continuations are replaced by $F'_i(x_i) = \overline{out_{F_i^*}}(x_i)$. These special continuations just send the received messages on the observable channels $out_{F_i^*}$. In [20] it is proved the following result:

Theorem 1. *Let S be a protocol which guarantees the well-formedness conditions* WFC1 *and* WFC2. *Then, S guarantees spi-authentication if and only if $S^{\mathbf{F}'}$ guarantees* NDC, *for all* $\mathbf{m}$.

An Example: The Wide Mouthed Frog Protocol. In this section we show how to use *NDC* to analyze a simplified version (also studied in [2]) of the Wide Mouthed Frog Protocol [9], we already discussed in Section 5.5. We still consider a version of the protocol with no time-stamps, which makes the protocol sensitive to a replay attack (as already remarked in [1]). Notice that, differently from Section 5.5, in the first message B is encrypted, as required by the original protocol.

$$\begin{array}{ll} \text{message 1} & A \rightarrow S : A, \{B, k_{AB}\}_{k_{AS}} \\ \text{message 2} & S \rightarrow B : \{A, k_{AB}\}_{k_{BS}} \\ \text{message 3} & A \rightarrow B : \{m_A\}_{k_{AB}} \end{array}$$

We specify two sessions of the protocol as the following CryptoSPA normal form[8]:

$$\begin{array}{rcl} A(m,k) & \stackrel{\text{def}}{=} & \overline{c_1}\,(A, \{(B,k)\}_{k_{AS}}) \,.\, \overline{c_3}\,\{m\}_k \\ B & \stackrel{\text{def}}{=} & c_2(y) \,.\, [\langle y, k_{BS}\rangle \vdash_{dec} z]\ [z \vdash_{snd} s]\ c_3(t) \,.\, [\langle t, s\rangle \vdash_{dec} w]\ \overline{p_F}\, w \\ S & \stackrel{\text{def}}{=} & c_1(u) \,.\, [\langle u\rangle \vdash_{snd} x]\ [\langle x, k_{AS}\rangle \vdash_{dec} y]\ [y \vdash_{snd} z]\ \overline{c_2}\,\{(A,z)\}_{k_{BS}} \,.\, S \\ P(m,m') & \stackrel{\text{def}}{=} & (A(m,k_{AB}) \,\|\, A(m',k'_{AB}) \,\|\, B \,\|\, B \,\|\, S \,\|\, p_F(z).F(z) \,\| \\ & & \|\, p_F(z).F(z)) \setminus p_F \end{array}$$

where c_1, c_2 and c_3 are the three channels over which messages 1,2 and 3 are communicated, respectively. Note that we have considered two instances of A and B in order to observe the replay attack. If we consider only one instance of A and B no attack is possible here. Note also that A has different messages and session keys in the two sessions.

Let us see if the protocol guarantees the well-formedness conditions. First, it is easy to see that $P(\mathbf{m}, \mathbf{m}')$ with no enemy, i.e., $P(\mathbf{m}, \mathbf{m}') \setminus \{c_1, c_2, c_3\}$, is trace equivalent to $F(\mathbf{m}) \,\|\, F(\mathbf{m}')$. This represents the intended execution of the protocol. As a matter of fact the two sessions can be executed in any possible interleaving. It is also easy to prove that $P_{spec}(\mathbf{m}, \mathbf{m}') \setminus \{c_1, c_2, c_3\} \approx_{trace} F(\mathbf{m}) \,\|\, F(\mathbf{m}')$. Thus, we finally have $P(\mathbf{m}, \mathbf{m}') \setminus C \approx_{trace} P_{spec}(\mathbf{m}, \mathbf{m}') \setminus C \approx_{trace} F(\mathbf{m}) \,\|\, F(\mathbf{m}')$.

Since $P(\mathbf{m}, \mathbf{m}')$ is well-formed, in order to check spi-authentication on it, we can verify if $P^{\mathbf{F}'}(\mathbf{m}, \mathbf{m}')$ guarantees *NDC*-authentication. Note that $P^{\mathbf{F}'}(\mathbf{m}, \mathbf{m}')$ is obtained from $P(\mathbf{m}, \mathbf{m}')$ by replacing $F(w)$ with $\overline{out_F}\, w$. Note also that in the two instances of B the continuation is exactly the same (this allows to model multiple sessions).

[8] This protocol specification is a bit simplified since there are only two users and the possible sessions are fixed in advance. However, this modelling is sufficient to show how the attack can be revealed.

We show that $P^{\boldsymbol{F'}}(\mathbf{m}, \mathbf{m'})$ does not guarantee such a property. Consider the enemy $X \stackrel{\text{def}}{=} c_2(x).\overline{c_2}\,x.\overline{c_2}\,x.c_3(y).\overline{c_3}\,y.\overline{c_3}\,y$. It is easy to see that $P \parallel_C X$ is able to execute the trace $\overline{out_F}\,\mathbf{m}.\overline{out_F}\,\mathbf{m}$ that is not a trace for process $\overline{out_F}\,\mathbf{m} \parallel \overline{out_F}\,\mathbf{m'}$. Hence, $P^{\boldsymbol{F'}}(\mathbf{m}, \mathbf{m'}) \parallel_C X \not\leq_{trace} \overline{out_F}\,\mathbf{m} \parallel \overline{out_F}\,\mathbf{m'} \approx_{trace} P^{\boldsymbol{F'}}(\mathbf{m}, \mathbf{m'}) \setminus C$ and NDC-authentication is not satisfied.

The enemy X intercepts messages 2 and 3 and replays them, inducing B to commit twice on message $\mathbf{m}$ (as shown by trace $\overline{out_F}\,\mathbf{m}.\overline{out_F}\,\mathbf{m}$). This attack is quite critical in some situations. As an example, $\mathbf{m}$ could be a request of money transfer that would be executed twice. In order to avoid this attack, it is possible to modify the protocol (as done in [2]) by adding nonce-based challenge responses.

6 Some Simple Comparison Results

In this section, we show that having a uniform treatment of security properties make it easier to study the relationships and the differences among them. First, we show that NDC may be seen as a sufficient condition for every property which is based on trace-preorder. This result is interesting since it relates the non-interference property NDC, originally proposed for modeling information flow security, to network security properties like, e.g., authentication. The result holds for what we will call *good candidates* for a pair of functions γ, α, i.e., processes P such that $\gamma(P) \setminus C \leq_{trace} \alpha(P)$. This condition is quite reasonable since we certainly want that at least the (decorated) protocol under no attacks, i.e., $\gamma(P) \setminus C$, "satisfies" $\alpha(P)$.

Proposition 13. *Let γ be a decoration function, let α be a function between processes and let P be a good candidate for γ, α, i.e., $\gamma(P) \setminus C \leq_{trace} \alpha(P)$. Then, $\gamma(P)$ is NDC implies that P is $GNDC^{\gamma,\alpha}_{\leq_{trace}}$.*

Proof. The fact that $\gamma(P)$ is NDC can be equivalentely expressed by the fact that P is $GNDC^{Id_C,\gamma(P)\setminus C}_{\approx_{trace}}$. Then, for every $X \in \mathcal{E}^{\phi}_{C}$ we have $Id_C(\gamma(P)) \parallel_C X \leq_{trace} \gamma(P)\setminus C$, i.e., $\gamma(P) \parallel_C X \leq_{trace} \gamma(P)\setminus C$. By the hypothesis that P is a good candidate, i.e., $\gamma(P)\setminus C \leq_{trace} \alpha(P)$, we obtain $\gamma(P) \parallel_C X \leq_{trace} \gamma(P)\setminus C \leq_{trace} \alpha(P)$ and so P is $GNDC^{\gamma,\alpha}_{\leq_{trace}}$. ∎

Note that if a pair γ, α does not have good candidates then it represents an empty property (no process satisfies it).

The result above shows that NDC, required on a process $\gamma(P)$, is stronger than any $GNDC^{\gamma,\alpha}_{\leq_{trace}}$ property, for all the good candidates P. For example, $\gamma_{Agree}(P) \in NDC$ implies $GNDC^{\gamma_{Agree},\alpha_{Agree}}_{\leq_{trace}}$ and $\gamma_{auth}(P) \in NDC$ implies $GNDC^{\gamma_{auth},\alpha_{auth^T_R}}_{\leq_{trace}}$, for their respective good candidates. This is quite intuitive since NDC basically requires that the protocol behaviour is completely preserved even under attack. So if $\gamma(P)$ satisfies a certain property under no attacks (i.e., it is a good candidate), the fact that $\gamma(P)$ is NDC will preserve such a property under every possible attack.

Example 11. Consider once more the protocol first presented in Example 1 written in the form required by the agreement property (Example 8).

$$\begin{aligned} P &= C(A,B,K_s) \parallel V^z(B,A) \\ C(a,b,k) &= \overline{c}\,\{k,a\}_K.\underline{0} \\ V^z(b,a) &= c(y).[\langle y,K\rangle \vdash_{dec} w][w \vdash_{fst} z].\underline{0} \end{aligned}$$

Recall that

$$\begin{aligned} \gamma_{Agree}(P) = {} & \overline{running\_ini}\,(A,B,K_s).\overline{c}\,\{K_s,A\}_K.\underline{0} \\ & \parallel\ c(y).[\langle y,K\rangle \vdash_{dec} w][w \vdash_{fst} z].\overline{commit\_res}\,(B,A,z) \end{aligned}$$

It is trivial to prove that P is a good candidate for $\gamma_{Agree}, \alpha_{Agree}$, i.e., $\gamma_{Agree}(P)\backslash C \leq_{trace} \alpha_{Agree}(P)$. As a matter of fact the only trace of $\gamma_{Agree}(P) \setminus C$ is $\overline{running\_ini}\,(A,B,K_s), \overline{commit\_res}\,(B,A,K_s)$. Thus, we can use NDC to check the agreement property, i.e., if we prove that $\gamma(P)$ is NDC then, by Proposition 13, we also have that $P \in GNDC^{\gamma_{Agree},\alpha_{Agree}}_{\leq_{trace}}$.

It can be checked (e.g., using the tools described in [15]) that $\gamma_{Agree}(P)$ is NDC whenever $K \notin \phi_I$. However, as noticed in Section 1.3, this protocol is flawed when more then one session is considered. As a consequence we have that $\gamma_{Agree}(P \parallel P)$ cannot be NDC (otherwise we would get a contradiction). As done in Example 8 we consider an enemy $X = c(w).\overline{c}\,(w).\overline{c}\,(w)$. It is easy to see that trace $\overline{running\_ini}\,(A,B,K_s).\overline{commit\_res}\,(B,A,K_s).\ \overline{commit\_res}\,(B,A,K_s)$ is in the set $Tr(\gamma_{Agree}(P \parallel P) \parallel_C X)$ but not in $Tr(\gamma_{Agree}(P \parallel P)\backslash C)$, as the second commit is not matched by any running action. So, as expected, $\gamma_{Agree}(P \parallel P)$ is not NDC. ■

In general, we observe that if $\lhd \subseteq \lhd'$ then $GNDC^{\gamma,\alpha}_{\lhd} \subseteq GNDC^{\gamma,\alpha}_{\lhd'}$, furthermore if for all $P \in \mathcal{E}$ we have $\alpha(P) \lhd \alpha'(P)$ and $\gamma'(P) \lhd \gamma(P)$ [9] then $GNDC^{\gamma,\alpha}_{\lhd} \subseteq GNDC^{\gamma',\alpha'}_{\lhd}$.

As an example, we study the natural extension of secrecy to two messages. Consider a protocol $P(m_1,m_2)$ and let us define two different γ decoration functions: γ^1 requiring the secrecy of m_1 and γ^2 requiring the secrecy of both the messages:

$$\begin{aligned} \gamma^1_{secret}(P(m_1,m_2)) &= P(m_1,m_2) \parallel KN(m_1) \\ \gamma^2_{secret}(P(m_1,m_2)) &= P(m_1,m_2) \parallel KN(m_1) \parallel KN(m_2) \end{aligned}$$

It is trivial to see that, for all $P(m_1,m_2)$ we have $\gamma^1_{secret}(P(m_1,m_2)) \leq_{trace} \gamma^2_{secret}(P(m_1,m_2))$. Thus we obtain $GNDC^{\gamma^2_{secret},\alpha_{secret}}_{\leq_{trace}} \subseteq GNDC^{\gamma^1_{secret},\alpha_{secret}}_{\leq_{trace}}$, reflecting the intuition that requiring the secrecy of both the messages is stronger than requiring the secrecy of the first one only.

We give another simple example, by considering message-oriented authentication. It is easy to prove that $\alpha_{auth^T_R}(P) \leq_{trace} \alpha_{auth^{T'}_{R'}}(P)$ whenever $T \geq T'$ and

[9] Notice that α and γ are counter-variant here.

$R \leq R'$. As a consequence $GNDC_{\leq_{trace}}^{\gamma_{auth},\alpha_{auth_R^T}} \subseteq GNDC_{\leq_{trace}}^{\gamma_{auth},\alpha_{auth_{R'}^{T'}}}$, reflecting the intuition that actions in T are executable only after at least one action in R has been executed. If we increment the set R, the number of processes satisfying message-oriented authentication increases. The same happens when we decrease set T, because the actions that are no more in T become direcly executable.

We end this section by showing how to apply the theory developed so far in order to check multiple properties with just one NDC check. We have observed that enlarging the decoration γ makes the resulting property stronger, i.e., if, for all P, $\gamma'(P) \lhd \gamma(P)$ then $GNDC_{\lhd}^{\gamma,\alpha} \subseteq GNDC_{\lhd}^{\gamma',\alpha}$. Thus, given two different properties characterized by the pairs (γ_1, α_1) and (γ_2, α_2), we can try to find a larger γ_3 that contains both the two decorations, i.e., such that $\gamma_1(P) \lhd \gamma_3(P)$ and $\gamma_2(P) \lhd \gamma_3(P)$, for all P. Intuitively, this corresponds to considering the sum of the two decorations. By Proposition 13 we directly obtain that if $\gamma_3(P) \setminus C \leq_{trace} \alpha_1(P)$ and $\gamma_3(P) \setminus C \leq_{trace} \alpha_2(P)$ then $\gamma_3(P)$ is NDC implies that P is $GNDC_{\leq_{trace}}^{\gamma_1,\alpha_1}$ and $GNDC_{\leq_{trace}}^{\gamma_2,\alpha_2}$. Thus, NDC may be used to simultaneously check both the two properties. Notice that, in general, the two conditions $\gamma_3(P) \setminus C \leq_{trace} \alpha_1(P)$ and $\gamma_3(P) \setminus C \leq_{trace} \alpha_2(P)$ are not restrictive, as $\alpha_1(P)$ and $\alpha_2(P)$ impose restrictions only on the actions added by their respective decorations. Thus, e.g., $\gamma_1(P) \setminus C \leq_{trace} \alpha_1(P)$ should imply that also $\gamma_3(P) \setminus C \leq_{trace} \alpha_1(P)$.

Example 12. Consider once more the protocol P discussed in Example 11 and let us write it as $P(K_s)$, i.e., parametric with respect to the session key.

$$\begin{aligned} P(K_s) &= C(A, B, K_s) \parallel V^z(B, A) \\ C(a, b, k) &= \overline{c}\,\{k, a\}_K.\underline{0} \\ V^z(b, a) &= c(y).[\langle y, K\rangle \vdash_{dec} w][w \vdash_{fst} z].\underline{0} \end{aligned}$$

Recall that

$$\begin{aligned} \gamma_{Agree}(P(K_s)) = \overline{running\_ini}\,(A, B, K_s).\overline{c}\,\{K_s, A\}_K.\underline{0} \\ \parallel\ c(y).[\langle y, K\rangle \vdash_{dec} w][w \vdash_{fst} z].\overline{commit\_res}\,(B, A, z) \end{aligned}$$

Suppose that we also want to check whether or not the secrecy of the session key is guaranteed. Recall that

$$\gamma_{secret}(P(K_s)) = P(K_s) \parallel KN(K_s)$$

Now if we simply consider the new decoration function

$$\gamma_{A\&S}(P(K_s)) = \gamma_{Agree}(P(K_s)) \parallel KN(K_s)$$

we obtain that $\gamma_{Agree}(P(K_s)) \leq_{trace} \gamma_{A\&S}(P(K_s))$ and $\gamma_{secret}(P(K_s)) \leq_{trace} \gamma_{A\&S}(P(K_s))$. Intuitively, $\gamma_{A\&S}(P(K_s))$ is the sum of the two decorations. Now, it is trivial to prove that $P(K_s)$ is still a good candidate for $\gamma_{A\&S}, \alpha_{Agree}$, i.e., $\gamma_{A\&S}(P) \setminus C \leq_{trace} \alpha_{Agree}(P)$. As a matter of fact the only trace of $\gamma_{A\&S}(P) \setminus C$ is $\overline{running\_ini}\,(A, B, K_s), \overline{commit\_res}\,(B, A, K_s)$. For the same reason, it is also

trivial to see that $P(K_s)$ is a good candidate for $\gamma_{A\&S}, \alpha_{secret}$. Thus, we can use NDC to simultaneously check agreement and secrecy. It can be checked (e.g., using the tools described in [15]) that $\gamma_{A\&S}(P(K_s))$ is NDC whenever $K \notin \phi_I$. Thus, for a single session, $P(K_s)$ guarantees both agreement and the secrecy of the session key. We have seen that for multiple sessions agreement is not guaranteed. As a matter of fact, trying to check NDC on $\gamma_{A\&S}(P(K_s) \parallel P(K'_s))$ fails because of the attack on authentication discussed in Example 11. ■

7 Conclusion

In this paper we have proposed a general scheme, called GNDC, that allows us to specify different security properties in a uniform setting. Rephrasing different properties in our uniform scheme is interesting from different perspectives: (*i*) it allows to better understand and compare the properties of interest, (*ii*) it allows to reuse general results and proof techniques, (*iii*) it allows to check all the rephrased properties as just one NDC check, (*iv*) since NDC is supported by the automated tools CVS/CoSeC [15], the GNDC scheme allows us to apply such tools to all the rephrased properties.

This analysis technique has been applied to many case studies, sometimes finding new failures or variant of known failures [14].

There are many extensions to the model presented in this paper that have already been developed. Extensions with time [21] allow us to detect attacks due to timing covert-channels and to analyze real-time cryptographic protocols [29]. Extensions with probabilities [3,4] allow us to refine process specification in order to detect flaws that are related to event probabilities. The extension of probabilistic models to deal with cryptographic protocols is currently under development. We are planning to compare our approach with other ones, based on different process calculi. Some preliminary results in this direction can be found in [30]. An alternative approach based on logical specification of the correct behavior and on partial model checking techniques has been proposed in [35,36,37].

References

1. M. Abadi and A. D. Gordon. Reasoning about cryptographic protocols in the spi calculus. In *Proceedings of CONCUR'97*, pages 59–73. Lecture Notes in Computer Science 1243, 1997.
2. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
3. A. Aldini, M. Bravetti, R. Gorrieri. A Process-algebraic Approach for the Analysis of Probabilistic Non-interference. To appear on *Journal of Computer Security*, 2003.
4. A. Aldini. Probabilistic Information Flow in a Process Algebra. In *12th Int. Conference on Concurrency Theory* (CONCUR'01), Springer LNCS 2154:152-168, Aalborg, Denmark, August 2001.

5. C. Bodei, P. Degano, R. Focardi, and C. Priami. Authentication via localized names. In *Proceedings of CSFW'99*, pages 98–110. IEEE press, 1999.
6. D. E. Bell, L. J. La Padula. Secure Computer Systems: Unified Exposition and Multics Interpretation. ESD-TR-75-306, MITRE MTR-2997, 1976.
7. M. Bugliesi, A. Ceccato, S. Rossi. Non-Interference Proof Techniques for the Analysis of Cryptographic Protocols. In *Proceedings of 2003 IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS Workshop on Issues in the Theory of Security* (WITS'03), April 5 - 6, 2003, Warsaw, Poland.
8. M. Bugliesi, A. Ceccato, and S. Rossi. Context-Sensitive Equivalences for Non-Interference based Protocol Analysis. In *Proc. of the 14th International Symposium on Fundamentals of Computation Theory*, FCT 2003, LNCS 2751, pag. 364-375, Springer-Verlag, 2003.
9. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London*, 426:233–271, 1989.
10. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0. http://www.cs.york.ac.uk/∼jac/papers/drareview.ps.gz, November 1997.
11. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
12. D. Dolev, A.C. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.
13. A. Durante, R. Focardi, and R. Gorrieri. CVS: A compiler for the analysis of cryptographic protocols. In *Proceedings of CSFW'99*, pages 203–212. IEEE press, 1999.
14. A. Durante, R. Focardi, and R. Gorrieri. CVS at Work: A Report on new Failures upon some Cryptographic Protocols. In *Procs. International Workshop Mathematical Methods, Models and Architectures for Computer Networks Security*, LNCS 2052, 287-299, Springer, St. Petersburg, Russia, 2001.
15. A. Durante, R. Focardi, and R. Gorrieri. A compiler for analysing cryptographic protocols using non-interference. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(4):488–528, October 2000.
16. R. Focardi, A. Ghelli, and R. Gorrieri. Using non interference for the analysis of security protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.
17. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1994/1995.
18. R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, 1997.
19. R. Focardi, R. Gorrieri. Classification of Security Properties. Part I: Information Flow. in *Foundations of Security Analysis and Design* (R.Focardi, R.Gorrieri eds), LNCS 2171, 331-396, Springer, 2001.
20. R. Focardi, R. Gorrieri, and F. Martinelli. A Comparison of Three Authentication Properties. *Theoretical Computer Science*, Volume 291(3), Pages 285-327, January 2003, Elsevier Science.
21. R. Focardi, R. Gorrieri and F. Martinelli. Real-Time Information Flow Analysis. *IEEE Journal on Selected Areas in Communications*. January 2003, Volume 21, Number 1. IEEE press.
22. R. Focardi, R. Gorrieri, and F. Martinelli. Message authentication through non-interference. In *Proceedings of 8th International Conference in Algebraic Methodology and Software Technology* (AMAST 2000), pages 258–272. Springer Lecture Notes in Computer Science 1816, 2000.

23. R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *proceedings of 27th International Colloquium on Automata, Languages and Programming (ICALP'00), (U. Montanari, ed.)*, pages 354–372. Lecture Notes in Computer Science 1853, July 2000.
24. R. Focardi, R. Gorrieri, and F. Martinelli. Secrecy in security protocols as non-interference. Workshop on secure architectures and information flow, ENTCS 32, 2000.
25. R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proceedings of World Congress on Formal Methods (FM'99)*, pages 794–813. Springer, Lecture Notes in Computer Science 1708, 1999.
26. J. A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 11–20. IEEE Press, 1982.
27. D. Gollman. What do we mean by entity authentication? In *Proceedings of Symposium in Research in Security and Privacy*, pages 46–54. IEEE Press, 1996.
28. D. Gollman. On the verification of cryptographic protocols - a tale of two committees. In *Workshop on secure architectures and information flow*, volume 32 of *ENTCS*, 2000.
29. R. Gorrieri, E. Locatelli, F. Martinelli. A Simple Language for Real-Time Cryptographic Protocol Analysis. In Proceedings of *12th European Symposium on Programming, (ESOP 2003)*, Springer LNCS 2618, pages 114-128, April 2003.
30. R. Gorrieri, F. Martinelli. Process Algebraic Frameworks for the Specification and Analysis of Cryptographic Protocols. In procs. 28th International Symposium on *Mathematical Foundations of Computer Science* (MFCS03), LNCS 2747, 46-67, Springer, Bratislava (SK), August 2003.
31. C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985.
32. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS'96*, pages 146–166. Lecture Notes in Computer Science 1055, 1996.
33. G. Lowe. A hierarchy of authentication specification. In *Proceedings of the 10th Computer Security Foundation Workshop*, pages 31–43. IEEE press, 1997.
34. W. Marrero, E. Clarke, and S. Jha. A model checker for authentication protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*. Rutgers University, Sep. 1997.
35. F. Martinelli. Analysis of security protocols as open systems. *Theoretical Computer Science* 290(1): 1057-1106 (2003)
36. F. Martinelli. Languages for description and analysis of authentication protocols. In *Proceedings of ICTCS'98*, pages 304–315. World Scientific, 1998.
37. F. Martinelli. Partial model checking and theorem proving for ensuring security properties. In *Proceedings of CSFW'98*, pages 44–52. IEEE press, 1998.
38. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
39. P. Y. A. Ryan and S. Schneider. Process algebra and non-interference. In *Proceedings of CSFW'99*, pages 214–227. IEEE press, 1999.
40. D. Sangiorgi. *Expressing Mobility in Process Algebra: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1992.
41. S. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of CSFW'98*, pages 54–65. IEEE Press, 1998.
42. S. Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9), September 1998.
43. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.

44. J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 144–161. IEEE Computer Society Press, 1990.
45. T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *Proceedings of the 1993 IEEE Computer Society Symposium on Security and Privacy (SSP '93)*, pages 178–195. IEEE Press, May 1993.
46. J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Proc. of Symposium in Research in Security and Privacy*, pages 55–61. IEEE Press, 1996.
47. J. Zhou and D. Gollmann. Towards verification of non-repudiation protocols. In *International Refinement Workshop and Formal Methods Pacific*, 1998.

Cryptographic Algorithms for Multimedia Traffic

Rosario Gennaro

I.B.M. T.J.Watson Research Center
P.O.Box 704, Yorktown Heights, NY 10598, USA
rosario@watson.ibm.com

Abstract. We survey several cryptographic algorithms that provide authentication and confidentiality for multimedia traffic over the Internet. We focus in particular on the problem of authenticating streams of data and on the problem of secure multicast, where streamed information is sent to a dynamic group of users.

1 Introduction

This paper describes various cryptographic algorithms for security of multimedia traffic over the Internet. There are several issues that are specific to multimedia data. Usually the data takes the form of *streams* rather than single message units. Also the data is usually delivered to large groups of users that might dynamically change. Efficiency and scalability are of utmost importance. These issues prevent a straightforward application of basic cryptographic tools for authentication and confidentiality and often require *ad hoc* solutions.

Streams Defined. A stream is a potentially very long (infinite) sequence of bits that a sender sends to a receiver. The stream is usually sent at a rate which is negotiated between the sender and the receiver or there may be a demand-response protocol in which the receiver repeatedly sends requests for additional (finite) amount of data. The main feature of streams which distinguish them from messages is that the receiver must consume the data it receives at more or less the input rate, i.e., it can't buffer large amounts of unconsumed data. In fact in many applications the receiver stores relatively very small amounts of the stream. In some cases the sender itself may not store the entire sequence, i.e., it may not store the information it has already sent out and it may not know anything about the stream much beyond of what it has sent out.

There are many examples of digital streams. Common examples include *digitized video* and *audio* which is now routinely transported over the Internet and also to television viewers via various means, e.g., via direct broadcast satellites and very shortly via cable, wireless cable, telephone lines etc. This includes both pre-recorded and stored audio/video programming as well as live feeds. Apart from audio/video, there are also *data feeds* (e.g., news feeds, stock market quotes etc.) which are best modeled as a stream.

R. Focardi and R. Gorrieri (Eds.): FOSAD 2001/2002, LNCS 2946, pp. 186–206, 2004.

Multicast Security. In multicast communication, messages are exchanged between members of a potentially dynamic group. In most applications these messages are usually streams. The basic secrecy problem is that messages should be visible only to legitimate members of the group (e.g. paying subscribers). Messages should also be authenticated so that receivers can make sure that they have not been tampered with. A more complex problem is that of *source authentication* which should allow the receivers to identify the sender of the message among the members of the group.

2 Cryptographic Preliminaries

In this section we recall some cryptographic concepts and terminology which will be useful later. For a good survey of basic cryptographic algorithms the reader is referred to [18]. In the following we denote with n the security parameter. We say that a function $\epsilon(n)$ is *negligible* if for all c, there exists an n_0 such that, for all $n > n_0$, $\epsilon(n) < 1/n^c$.

Collision-Resistant Hash Functions. Let $\mathcal{H}$ be a family of functions that map arbitrarily long binary strings into binary strings of a fixed length k. We say that $\mathcal{H}$ is a *collision-resistant* family of hash functions if any polynomial time algorithm who is given as input a description of a random element $H \in \mathcal{H}$, finds a collision, i.e., a pair (x, y) such that $x \neq y$ and $H(x) = H(y)$, only with negligible probability $\epsilon(k)$.

SHA-1 [15] is a conjectured collision-resistant hash functions, i.e., it can be thought as a random representative of a family $\mathcal{H}$ with the above property.

Signature Schemes. A *signature scheme* is a triplet (G, S, V) of probabilistic polynomial-time algorithms satisfying the following properties:

- G is the *key generation* algorithm. On input 1^n it outputs a pair $(SK, PK) \in \{0,1\}^{2n}$. SK is called the secret (signing) key and PK is called the public (verification) key.
- S is the *signing* algorithm. On input a message M and the secret key SK, it outputs a signature σ.
- V is the *verification* algorithm. For every $(PK, SK) = G(1^n)$ and $\sigma = S(SK, M)$, it holds that $V(PK, \sigma, M) = 1$.

In [10] security for signature schemes is defined in several variants. The strongest variant is called "existential unforgeability against adaptively chosen message attack". That is, we require that no efficient algorithm will be able to produce a valid signed message, even after seeing several signed messages of its choice.

One-Time Signatures. A special kind of signature schemes satisfy the [10] definition of security only if we allow the adversary to see a limited number of signed messages. In particular there exists signature schemes that are secure only if used to sign a *single* message. The main advantage of this type of schemes is that they are usually much faster to execute than regular signature schemes.

Message Authentication Codes. A *message authentication code (MAC)* is a pair $(Auth, Ver)$ of probabilistic polynomial-time algorithms satisfying the following properties:

- $Auth$ is the *authenticating* algorithm. On input a message M and the secret key $K \in \{0,1\}^n$, it outputs a tag τ.
- Ver is the *verification* algorithm. For every $K \in \{0,1\}^n$ and $\tau = Auth(K, M)$, it holds that $Ver(K, \tau, M) = 1$.

Security for MACs can be defined analogously to signature schemes. That is, we require that no efficient algorithm will be able to produce a valid authenticated message, even after seeing several signed messages of its choice, without knowing the key K which is selected at random from $\{0,1\}^n$.

The main difference between MACs and Signature Schemes is that the latter provide *non-repudiation*, i.e. the receiver can prove to a third party that the message originated with the sender (the owner of the secret key). This is not possible for MACs since both sender and receiver share the secret key. MACs are usually much more efficient algorithms than digital signatures.

Public-Key Encryption. A *public key encryption scheme* is a triplet (G, E, D) of probabilistic polynomial-time algorithms satisfying the following properties:

- G is the *key generation* algorithm. On input 1^n it outputs a pair $(SK, PK) \in \{0,1\}^{2n}$. SK is called the secret (decryption) key and PK is called the public (encryption) key.
- E is the *encryption* algorithm. On input a message M and the public key PK, it outputs a ciphertext c.
- D is the *decryption* algorithm. For every $(PK, SK) = G(1^n)$ and $c = E(PK, M)$, it holds that $D(SK, c) = M$.

There are several levels of security for encryption schemes, depending if the attacker is active or passive. For the purpose of this paper we will stick to the lowest level of security: *indistinguishability*. The requirement is that the ciphertext space of a message M should be computationally indistinguishable from the ciphertext space of a different message M'. Basically it requires a randomized encryption algorithm so that given a ciphertext c and a candidate message M, the attacker cannot verify if c is an encryption of M or not.

Private-Key Encryption. A *private-key encryption* scheme is a pair (Enc, Dec) of probabilistic polynomial-time algorithms satisfying the following properties:

- Enc is the *encryption* algorithm. On input a message M and the secret key $K \in \{0,1\}^n$, it outputs a ciphertext c.
- Dec is the *decryption* algorithm. For every $K \in \{0,1\}^n$ and $c = Enc(K, M)$, it holds that $Dec(K, c) = M$.

Security for private-key encryption schemes can be defined analogously to public-key schemes. Private-key encryption schemes are more efficient in practice, but they require sender and receiver to agree on a secret key in advance.

Commitment Schemes. Commitment schemes simulate the role of *envelopes*. A commitment scheme is a protocol between a Sender and a Receiver. In a first phase, the Sender who commits to a message m, and the Receiver engage in an exchange of messages, at the end of which the Receiver has no idea what m is. This first phase is followed by a revealing phase in which the Sender *opens* the commitment to produce a decommitment string d, which includes the message m and a sort of "proof" that m is the correct value. There should be a unique way in which a Sender can open a commitment (i.e. the first phase *committed* the Sender to m).

We limit ourselves to non-interactive commitment schemes, i.e. those in which commitment and reveal phase consist of a single message from the Sender to the Receiver. Thus a commitment scheme consists of a pair of probabilistic polynomial-time algorithms $Comm, Decomm$ such that:
(1) $Decomm(Comm(m)) = m$;
(2) the random variable $Comm(m)$ should be computationally indistinguishable $Comm(m')$ for a different message m';
(3) it is computationally infeasible to generate a commitment string c and two valid decommit strings $d \neq d'$ such that d and d' include respectively different messages $m \neq m'$.

3 How to Authenticate or Sign Digital Streams

If we go back to the definition of streams, it becomes clear that message-oriented signature schemes cannot be directly used to sign streams since the receiver cannot be expected to receive the entire stream before verifying the signature. If a stream is infinitely long (e.g., the 24-hours news channel), then it is impossible for the receiver to receive the entire stream and even if a stream is finite but long the receiver would have to violate the constraint that the stream needs to be consumed at roughly the input rate and without delay.

3.1 Simple Authentication of Streams

Notice that if the receiver were only interested in establishing the identity of the sender, a solution based on MACs would suffice. Indeed, once the sender and receiver share a secret key, the stream could be authenticated block by block using a MAC computation on it. Since MACs are usually faster than signatures to compute and verify, this solution would fit the bill in terms of efficiency as well.

However, a MAC-based approach would not enjoy the non-repudiation property. In this section we are going to describe a scheme which achieves non-repudiation. In order for this property to be meaningful in the context of streams we need to require that *each* prefix of the stream to be non-repudiable. That is, if the stream is $\mathcal{B} = B_1, B_2, \ldots$ where each B_i is a block, we require that each prefix $\mathcal{B}_i = B_1 \ldots B_i$ be non-repudiable. This rules out a solution in which the sender just attaches a MAC to each block and then signs the whole stream at the end.

This is to prevent the sender from interrupting the transmission of the stream before the non-repudiability property is achieved. Also it is a guarantee for the receiver. Consider indeed the following scenario: the receiver notices that the stream she is downloading is producing damages to her machine (streams can also model Java applets). She interrupts the transfer in order to limit the damage, but at the same time she still wants some proof to bring to court that the substream downloaded so far did indeed come from the sender.

In general non-repudiation is crucial when the stream is being sold as an electronic merchandise. With the advent of music and video distribution over the internet, it is clear that such transactions must be protected with mechanisms that allow the resolution of disputes through non-repudiation.

Later on we will also see that in the context of secure multicast a solution based on digital signatures could be preferable to one based on MACs. Indeed in that scenario (even if non-repudiation is not required) to simply sign the content may end up being the simplest and most efficient solution, since it avoids problems of key management among a large number of users.

3.2 Non-optimal Solutions for Signing of Streams

We first describe some solutions which are lacking optimality in various ways.

One type of solution splits the stream in blocks. The sender signs each individual block and the receiver loads an entire block and verifies its signature before consuming it. This solution also works if the stream is infinite. However, this solution forces the sender to generate a signature for each block of the stream and the receiver to verify a signature for each block. With today's signature schemes either one or both of these operations can be very expensive computationally. Which in turns means that the operations of signing and verifying can create a bottleneck to the transmission rate of the stream.

Another type of solution works only for finite streams which are known in advance. In this case, once again the stream is split into blocks. Instead of signing each block, the sender creates a table listing cryptographic hashes of each of the blocks and signs this table. When the receiver asks for the authenticated stream, the sender first sends the signed table followed by the stream. The receiver first receives and stores this table and verifies the signature on it. If the signature matches, then the receiver has the authenticated cryptographic hash of each of blocks in the stream and thus each block can be verified when it arrives. The problem with this solution is that it requires the storage and maintenance of a potentially very large table on the receiver's end. In many realistic scenarios the receiver buffer is very limited compared to the size of the stream, (e.g., in MPEG a typical movie may be 20 GBytes whereas the receiver buffer is only required to be around 250Kbytes). Therefore the hash table can itself become fairly large (e.g., 50000 entries in this case or 800Kbytes for the MD5 hash function) and it may not be possible to store the hash table itself. Also, the hash table itself needs to be transmitted first and if it is too large then there will be a significant delay before the first piece of the stream is received and consumed. To address the problem of large tables one can also come up with a hybrid scheme in which

the stream is split in consecutive pieces and each piece is preceded by a small signed table of contents.

The above solution can be further modified by using an authentication tree: the blocks are placed as the leaves of a binary tree and each internal node takes as a value the hash of its children (see [13].) This way the sender needs to sign and send only the root of this tree. However, in order to authenticate each following block the sender has to send the whole authentication path (i.e., the nodes on the path from the root to the block, plus their siblings) to the receiver. This means that if the stream has k blocks, the authentication information associated with each block will be $O(\log k)$.

The solution described next eliminates all these shortcomings. The basic idea works for both infinite and finite streams, only one expensive digital signature is ever computed, there are no big tables to store, and the size of the authentication information associated with each block does not depend on the size of the stream.

3.3 An Optimal Solution to Sign Streams

In order to describe this optimal solution we make some reasonable/practical assumptions about the nature of the streams being authenticated. First of all we assume that it is possible for the sender to embed authentication information in the stream. This is usually the case in most real-world situations like MPEG video/audio. We also assume that the receiver has a "small" buffer in which it can first authenticate the received bits before consuming them. Finally we assume that the receiver has processing power or hardware that can compute a small number of fast cryptographic checksums faster than the incoming stream rate while still being able to play the stream in real-time.

The basic idea is to divide the stream into blocks and embed some authentication information in the stream itself. The authentication information of the i^{th} block will be used to authenticate the $(i+1)^{st}$ block. This way the signer needs to sign just the first block and then the properties of this single signature will "propagate" to the rest of the stream through the authentication information. Of course the key problem is to perform the authentication of the internal blocks fast. We distinguish two cases: if the stream is finite or infinite.

Finite Streams. In the first scenario the stream is finite and is known in its entirety to the signer in advance. This is not a very limiting requirement since it covers most of the Internet applications (digital movies, digital sounds, applets). In this case we will show that a *single* hash computation will suffice to authenticate the internal blocks. The idea is to embed in the current block a hash of the following block (which in turns includes the hash of the following one and so on...)

Assume for simplicity that the stream is such that it is possible to reserve 20 bytes of extra authentication information in a block of size c. The stream is logically divided into blocks of size c. The receiver has a buffer of size c. The receiver first receives the signature on the 20 byte hash (e.g., SHA-1) of the first block. After verification of the signature the receiver knows what the hash of the first block should be and then starts receiving the full stream and starts

computing its hash block by block. When the receiver receives the first block, it checks its hash against what the signature was verified upon. If it matches, it plays the block otherwise it rejects it and stops playing the stream. How are other blocks authenticated? The key point is that the first block contains the 20 byte hash of the second block, the second block contains the 20 byte hash of the third block and so on... Thus, after the first signature check, there are just hashes to be checked for every subsequent block.

In more detail, let (G, S, V) be a regular signature scheme. The sender has a pair of secret-public key $(SK, PK) = G(1^n)$ of such signature scheme. Also let H be a collision-resistant cryptographic hash function. If the original stream is

$$\mathcal{B} = B_1, B_2, \ldots, B_k$$

and the resulting signed stream is

$$\mathcal{B}' = B'_0, B'_1, B'_2, \ldots, B'_k$$

the processing is done *backwards* on the original stream as follows:

$$B'_k =< B_k, 00 \ldots 0 >$$

$$B'_i =< B_i, H(B'_{i+1}) > \text{ for } i = 1, \ldots, k-1$$

$$B'_0 =< H(B'_1, k), S(SK, H(B'_1, k)) >$$

Notice that on the sender side, computing the signature and embedding the hashes requires a single *backwards* pass on the stream, hence the restriction that the stream is fully known in advance. Notice also that the first block B'_0 of the signed stream contains an encoding of the length of the stream (k).

The receiver verifies the signed stream as follows: on receiving $B'_0 =< B, A_0 >$ she checks that

$$V(PK, A_0, B) = 1$$

and extracts the length k in blocks of the stream (which we may assume is encoded in the first block). Then on receiving $B'_i =< B_i, A_i >$ (for $1 \leq i \leq k$) the receiver accepts B_i if

$$H(B'_i) = A_{i-1}$$

Thus the receiver has to compute a single public-key operation at the beginning, and then only one hash evaluation per block. Notice that no big table is needed in memory.

Infinite Streams. The second case is for (potentially infinite) streams which are not known in advance to the signer (for example live feeds, like sports event broadcasting and chat rooms). In this scenario it is important that the operation of signing (and not just verification) also be fast, since the sender himself is bound to produce an authenticated stream at a potentially high rate.

The solution in this case is more complicated as it requires several hash computations to authenticate a block (although depending on the embedding mechanism these hash computations can be amortized over the length of the

block). The size of the embedded authentication information is also an issue in this case. The idea here is to use fast 1-time signature schemes (introduced in [11,12]) to authenticate the internal blocks. So block i will contain a 1-time public key and also the 1-time signature of itself with respect to the key contained in block $i-1$. This signature authenticates not only the stream block but also the 1-time key attached to it.

More in detail: let us denote with (G, S, V) a regular signature scheme and with (g, s, v) a 1-time signature scheme. With H we still denote a collision-resistant hash function. The sender has long-lived keys $(SK, PK) = G(1^n)$. Let

$$\mathcal{B} = B_1, B_2, \ldots$$

be the original stream (notice that in this case we are not assuming the stream to be finite) and

$$\mathcal{B}' = B'_0, B'_1, B'_2, \ldots$$

the signed stream constructed as follows. For each $i \geq 1$ let us denote with $(sk_i, pk_i) = g(1^n)$ the output of an independent run of algorithm g. Then

$$B'_0 =< pk_0, S(SK, pk_0) >$$

(public keys of 1-time signature schemes are usually short so they need not to be hashed before signing)

$$B'_i =< B_i, pk_i, s(sk_{i-1}, H(B_i, pk_i)) > \text{ for } i \geq 1$$

Notice that apart from a regular signature on the first block, all the following signatures are 1-time ones, thus much faster to compute (including the key generation, which does not have to be done on the fly.)

The receiver verifies the signed stream as follows. On receiving $B'_0 =< pk_0, A_0 >$ she checks that

$$V(PK, A_0, pk_0) = 1$$

and then on receiving $B'_i =< B_i, pk_{i+1}, A_i >$ she checks that

$$v(pk_{i-1}, A_i, H(B_i, pk_i)) = 1$$

whenever one of these checks fails, the receiver stops playing the stream. Thus the receiver has to compute a single public-key operation at the beginning, and then only one 1-time signature verification per block.

Hybrid Schemes. In the on-line scheme, the length of the embedded authentication information is of concern as it could cut into the throughput of the stream. In order to reduce it, hybrid schemes can be considered. In this case we assume that some asynchrony between the sender and receiver is acceptable.

Suppose the sender can process a group (say 20) of stream blocks at a time before sending them. With a pipelined process this would only add an initial delay before the stream gets transmitted. The sender will sign with a one-time key only 1 block out of 20. The 20 blocks in between these two signed blocks will be authenticated using the off-line scheme. This way the long 1-time signatures and the verification time can be amortized over the 20 blocks.

A useful feature of various proposed 1-time signature scheme is that it allows the verification of (the hash of) a message bit by bit. This allows to actually "spread out" the signature bits and the verification time among the 20 blocks. Indeed, if we assume that the receiver is allowed to play at most 20 blocks of unauthenticated information before stopping if tampering is detected we can do the following. We can distribute the signature bits among the 20 blocks and verify the hash of the first block bit by bit as the signature bits arrive. This maintains the stream rate stable since we do not have long signatures sent in a single block and verification now takes 3-4 hash computations per block, on *every* block.

The biggest drawback of the schemes we presented is that they are not very resilient to the problem of packet losses. Indeed if a block is missing then the authentication chain is broken and the rest of the scheme cannot be authenticated. An hybrid scheme helps in this case as well since the chain can be restarted when the next signed packet arrives.

3.4 Bibliographical Note

The scheme for stream signing described above was presented in [9]. The paper contains full proofs of security by reduction: i.e. it is shown that an attacker that is able to forge a signed stream in the schemes above must necessarily be able to either break the underlying signature schemes (either the long-lived or the one-time scheme in the on-line solution) or the collision-resistant hash function. The paper also discusses issues related to the choices of parameters and algorithms. In particular the choice of one-time signatures in the on-line case is important since the size of the signatures and keys could become problematic.

A different kind of efficient signature schemes, valid for a small number of messages, rather than just one is discussed in [17]. The paper also discusses their application to stream signing.

Some techniques to deal with the issue of packet loss are described in the original paper [9]. A modification of the basic scheme which is highly tolerant to packet loss is presented in [16].

4 Authentication in Secure Multicast

In Multicast Communication there is a dynamic group of users who exchange messages. These are usually in the form of streams (video, audio, stock quotes, teleconferencing etc.) There are two basic authentication problems in secure multicast: message and source authentication.

Message Authentication: Here the problem is to make sure that the message was not tampered with during transit. This problem is easily solvable by using MAC algorithms. The users in the group all share a key which is used to compute and verify MACs. When a message is received and verified, the users know that it was sent by a member of the group and it was not modified.

Source Authentication: Here the problem is to identify who sent the message inside the group. Here basic MACs do not work anymore since the receiver can also impersonate the user. Notice that the property we require is weaker than non-repudiation. Indeed we are only asking that a member of the group can identify the source of the message, but not necessarily be able to *prove* to a third party that this is the case. In the following we will focus only on the source authentication problem.

Clearly the previous solution for stream signing works here as well. Each member of the group holds a public key and uses the previously described algorithms to sign the streamed messages. However, it is a little bit of an overkill, especially in the on-line case, since we are doing more work than it's really required by the problem. In this section we will show some algorithms for source authentication which are more efficient than full stream signing. These algorithms can be considered something of a hybrid between signatures and MACs.

When we talk about efficiency, we stress that we are talking about the following important parameters:

- Work overhead: how much computation is required per packet to authenticate and verify;
- Bandwidth: how much packet expansion is required to include authentication data;
- Secure Memory: key length at sender and receiver.

4.1 Timed Authentication

This solution uses public-key techniques but in a limited way. It does not achieve non-repudiation, even if it uses digital signatures.

A sender in the group has a key pair PK, SK used to compute signatures under the signature scheme G, S, V. Also let $Comm, Decomm$ and $Auth, Ver$ be respectively a commitment scheme and a MAC scheme known to all members of the group. If the original stream is

$$\mathcal{B} = B_1, B_2, B_3 \ldots$$

then the resulting signed stream is

$$\mathcal{B}' = B'_1, B'_2, B'_3, \ldots$$

computed as follows:

$$B'_1 =< B_1, Comm(K_1), S(PK, B_1 \circ Comm(K_1)) >$$

$$B'_2 =< B_2, Comm(K_2), MAC(K_1, B_2 \circ Comm(K_2)) >$$

$$B'_3 =< B_3, Comm(K_3), Decomm(K_1), MAC(K_2, B_3 \circ Comm(K_3)) >$$

and following that

$$B'_i =< B_i, Comm(K_i), Decomm(K_{i-2}), MAC(K_{i-1}, B_i \circ Comm(K_i)) >$$

Notice that on the sender side, apart from the first packet, computing the authentication data requires only a MAC, a commitment and a decommitment step, which are all quite fast computations.

Verification proceeds as follows. B'_1 can be authenticated upon receipt since it contains a digital signature of its content. B'_2 will be authenticated when B'_3 arrives. The procedure is the following: take K_1 out of B'_3 and verify that it is a proper decommitment of the value contained in B'_1. Since B'_1 is already authenticated this will authenticate the value K_1. Now use K_1 to verify the MAC on B'_2. The verification proceeds similarly for all the following blocks.

It should be clear why the scheme does not achieve non-repudiation. Once the MAC keys are known, the receiver can modify the previous blocks (with the respective MACs).

By the same reasoning we have the following attack: if the receiver gets B'_{i+1} before B'_i then it cannot trust any of the following packets, since B'_{i+1} may come from an adversary who has already seen the key. There are two possible solutions to this problem.

Two-way Acknowledgments. The simplest thing to do is for the sender to wait to send B'_{i+1} until the receiver acknowledge receipt of B'_i. This solution is however quite cumbersome in multicast scenarios where the size of the recipient group might be quite large.

Timing Assumption. Assume that sender and receiver synchronize clocks with the first block. And also assume that the two clocks do not drift too much apart (i.e. we have an upper bound δ_t on the drift after t time units from synchronization). Then the receiver accepts block B'_i if

$$ArrivalTime_i + \delta_t < DepartureTime_{i+1}$$

and the departure time can be estimated as

$$DepartureTime_i = DepartureTime_0 + \frac{i}{r}$$

where r is the sending rate.

The main problem with the above approach is that the sending rate must be slower than possible network delays, which may be a severe limitation in some applications. Moreover this basic scheme does not tolerate packet loss since as soon as a block is missing the whole authentication chain is broken. Notice that a block might be missing not just because the network dropped it, but also because it might have been rejected by the receiver because of the timing rule.

Dealing with Packet Loss. We can solve this problem by using a specific implementation of the commitment function. Let F be a pseudo-random function which is also collision-resistant (several conjectured examples exist in the literature, basically any family of collision-resistant hash function can be also considered a family of PRF functions). Then we define

$$K_i = F^{n-i}(K_0)$$

where K_0 is a random initial value. We also commit to $K_1 = Fn - 1(K_0)$ in the usual manner[1]. Now if a packet is lost we can still authenticate the following keys, by iterating the function F enough times to get to the first authenticated key that was not lost.

Improving Transfer Rate. There are several possible approaches to this problem. Rather than sending the key in the next packet we wait for d packets. This increases the time from which the packet is received to when it's authenticated.

Another possible approach is to schedule the revealing time of the key, on a time-interval basis rather than on a number of packets basis. This allows the sender to accommodate variable transfer rates.

Finally if we embed more than one authentication chain in the stream, we can accommodate different receivers with varying receiving rates.

4.2 A MAC-Only Solution

It would seem that the requirement for source authentication in multicast groups require the use of public-key solutions. Yet in this section we will show that if we relax the notion of security a little bit, it is possible to achieve source authentication by using only MACs.

Clearly a MAC-only solution could be obtained by having each pair of group users exchange a key, and a sender would MAC each message N times (where N is the number of users), once with each key. Clearly this does not scale well for large groups.

The scheme we are going to describe uses a different idea. The sender holds a set of ℓ keys. Each receiver holds a subset of these keys. Clearly if a group of receivers covers the whole set of keys of the sender they could easily impersonate him. Thus we relax our security definition by asking that any coalition of less than w receivers cannot impersonate the sender. The parameter w is called the *impersonation threshold* of the system.

Let $\mathcal{R} = \{K_1, K_2, \ldots, K_\ell\}$ be the keys held by the sender. A user u is given a subset $\mathcal{R}_u \subset \mathcal{R}$ of the keys. Each key K_i is included in $\mathcal{R}_u$ with probability $\frac{1}{w+1}$, independently for every i and every u.

The sender sends a message M and attaches ℓ MACs, one with each key he holds. The user u accepts if *all* the MACs created with the keys in $\mathcal{R}_u$ are correct.

Now that we specified the scheme, we only need to pinpoint a sufficiently large value for ℓ. In [4] the following theorem is proved, by a simple probability argument.

Theorem 1. *If $\ell = e(w+1)\ln\frac{1}{q}$ then the probability that a coalition of w users can impersonate the sender to a specific user u is less than q.*

[1] To save on code we can also use F, for the commitment purpose i.e. $Comm(K_1) = F^n(K_0)$.

Theorem 1 holds with respect to a specific subset of w parties. We want security with respect to *any* coalition. If we fix

$$q = \left[n \cdot \binom{n}{w} \right]^{-1}$$

(i.e. one over the number of all pairs of one player and w-coalitions), then a probabilistic argument tells us that there exists a key assignment for which *no* $\mathcal{R}_u$ is covered by a w-coalition. The proof is not really constructive, but it is possible to test key assignments until a correct one is found.

Notice that the total number of keys is $\approx e(w+1)^2 \ln N$, while each player holds $\approx e(w+1)\ln N$ keys.

Reducing the Bandwidth. We can reduce the communication overhead by considering shorter (thus weaker) MACs but more keys. At the extreme the MACs could be one-bit long (thus forgeable with probability 1/2).

For example if we set $\ell = 4e(w+1)\ln\frac{1}{q}$, with one-bit MACs, by repeating the probabilistic argument mentioned above it is possible to claim that with high probability a coalition of w players does not know up to $\log\frac{1}{q}$ keys of a receiver. Before we could only prove that the coalition missed at least *one* key.

Since one-bit MACs are forgeable with probability 1/2, the probability of fooling the user is at most q.

4.3 Bibliographical Note

The timed authentication scheme described in Section 4.1 was presented independently in several papers [1,2,5,16].

The MAC-only solution described in Section 4.2 originates from [4]. The reader is referred to the full paper for a more detailed description of the solution, including the case of multiple senders.

5 Secrecy in Secure Multicast

Protecting the confidentiality of multicast communication is easier, especially if the group is static. The members of the group all share the same key for a symmetric encryption scheme which is used to encrypt the data.

Technically, things become more interesting when the group is dynamic. User addition is easily handled by communicating the common key directly to the new user. If previous communications should be kept secret from the new user, a new key can be generated for the whole group.

User deletion or *revocation* is the most problematic issue. We are going to deal with this problem for the rest of the section. In the literature this problem has been referred also as *Broadcast Encryption*.

5.1 A Simple Tree Scheme

Let $u_0, \ldots, u_{N-1}$ be the users. Think of them as leaves in a binary tree. In the following we will denote with $p(v)$ and $s(v)$ respectively the parent and the sibling of a node v.

Each node v in the tree is associated with a key K_v. Each user receives the keys on the path from its leave to the root. The root key is used to encrypt traffic, since every user knows it.

If a user u is removed, new keys K'_v are generated for the nodes in the path from u to the root. The new keys are communicated as follows.

- $K'_{p(u)}$ is encrypted with $K_{s(u)}$;
- for all the other nodes v in the path, $K'_{p(v)}$ is encrypted with K'_v and $K_{s(v)}$.

It should be clear that the new keys can be decrypted by (and only by) all the users that are supposed to know them.

Each user holds $\log N + 1$ keys and the revocation of a user creates a message of $2 \log N - 1$ ciphertexts.

It is possible to improve the communication overhead by a factor of 2, by using pseudorandom number generators. Let G be a PRG that doubles its input $G(x) = L(x) \circ R(x)$. When u is removed the new keys are computed using some random seeds r's. The new key $K'_v = L(r_v)$ where the seeds are set as follows:

- $r_{p(u)}$ is set to a random value r.
- for all the other nodes v in the path, $r_{p(v)}$ is set to $R(r_v)$.

Each value r_v is encrypted with $K_{s(v)}$ and clearly from r_v one can compute $K'_v, K'_{p(v)}, \ldots, K'_{root}$.

5.2 Stateless Receivers

In the previous scheme we change the keys continously as the group changes. We expect the group members to update their internal state with the new keys. A large class of applications however deal with *stateless receivers*, i.e. devices that cannot update their internal state. Thus these devices are initialized with some long-lived keys which must be able to decrypt the data for any group configuration in which the device is enabled. In this section we present a general framework called *Subset Cover* for stateless receivers.

An algorithm defines k subsets $S_1, \ldots, S_k$ of the set of group members U . Each subset S_i is associated with a long-lived key L_i which does *not* change over time. If the user $u \in S_i$ then u must know L_i.

When a subset R of members must be revoked, the set N/R is partitioned into disjoint subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_m}$. The session key K (which is changed continously) is encrypted with keys $L_{i_1}, L_{i_2}, \ldots, L_{i_m}$. The ciphertext is

$$< i_1, \ldots, i_m, E(L_{i_1}, K), \ldots, E(L_{i_m}, K), E'(K, M) >$$

where E, E' are symmetric encryption schemes.

Decryption is performed by having user u determining for which i_j $u \in S_{i_j}$ and then decrypt the session key using the correct key L_{i_j}.

Thus a specific algorithm in this framework is determined by

1. the collection of subsets S_i;
2. the assignment of key L_i to subset S_i;
3. a method to cover N/R by disjoint subsets of the S_i's;
4. a method for user u to find its *cover* S_i.

The parameters to keep in mind are the following: (a) message length, (b) storage size at the receiver's end, (c) message processing time at the receiver's end. These are functions of $n = |U|$ the size of the universe of members, and $r = |R|$ the size of the revoked group.

We present two instantiations of the framework:

- **Complete Subtree:** which has parameters (a) $r \log \frac{n}{r}$, (b) $\log n$, (c) $O(\log \log n)$.
- **Subset Difference:** which has parameters (a) $2r$, (b) $\frac{\log^2 n}{2}$, (c) $O(\log n)$.

The Complete Subtree Method. This algorithm can be considered a generalization of the previous tree algorithm to the case of stateless receivers.

Think of users as the n leaves of a binary tree. The collection of sets S_i in this method consists of all the *complete* subtrees of this full tree. An user $u \in S_i$ if $\mathsf{root}(S_i)$ is an ancestor of u.

Each subset S_i is assigned a random key L_i which is given to all the users in the set.

Now suppose we want to remove r users $R = \{u_1, \ldots, u_r\}$. Consider the minimal subtree $T(R)$ that contains such users and the root. The cover is composed by all the subsets $S_{i_1}, \ldots, S_{i_m}$ that "hang off" $T(R)$. In other words all the subtrees whose root is adjacent to nodes of outdegree 1 in $T(R)$ (and do not belong to $T(R)$). Clearly $S_{i_1}, \ldots, S_{i_m}$ is a partition of U/R.

How big is m? An easy upper bound on m is $r \log n$, since there are at most r paths from the root to a leave in $T(R)$ and each path has length $\log n$. An easy induction argument actually improves this to $r \log \frac{n}{r}$.

Clearly each user belongs to $\log n$ sets, so it has to store that many keys.

Finally a user u belongs to the subtree S_{i_j} in the cover, if and only if u and $\mathsf{root}(S_{i_j})$ have the longest common prefix. This can be computed in $O(\log \log n)$ time if given the appropriate preprocessing.

If one compares this scheme to the basic tree algorithm, it can be seen that this scheme requires a *single* decryption, while the other one requires $O(\log n)$ decryptions, when users are revoked.

The Subset Difference Method. In this scheme we totally remove the dependency on n from the message expansion parameter (n may grow a lot). The tradeoff is that we square the number of keys held by each player.

Again, think of the n users as leaves of a full binary tree. Each subset S_{ij} can be described as a subtree minus another subtree. More specifically a subset S_{ij} is associated with two nodes v_i and v_j in the tree, such that v_i is an ancestor of v_j. The users in S_{ij} are the users in the subtree rooted at v_i minus the ones in the subtree rooted at v_j.

Suppose we want to revoke r users $R = \{u_1, \ldots, u_r\}$. We need to find a cover $S_{i_1 j_1}, \ldots, S_{i_m j_m}$ which partitions U/R. As before we consider the minimal subtree $T(R)$ that contains such users and the root. The cover is found via an iterative procedure which looks for maximal chains of nodes with outdegree 1 in $T(R)$. More specifically, such a chain $[v_{i_1}, \ldots, v_{i_\ell}]$ is such that: (1) $v_{i\ell}$ is either a leaf or a node of outdegree 2; (2) the parent of v_{i_1} is either the root or a node of outdegree 2; (3) each intermediate node $v_{i_1}, \ldots, v_{i_{\ell-1}}$ has outdegree 1. Note that all nodes of outdegree 1 in $T(R)$ belong to precisely one of such chains.

For each such chain, where $\ell \geq 2$, the procedure adds the subset S_{i_1, i_ℓ} to the cover. It is not hard to see that the cover size is at most $2r - 1$.

If we assign a random key L_{ij} to each subset S_{ij}, it would result in $O(N)$ keys held by each player. Indeed consider each complete subtree T_k such that $u \in T_k$. For each sub-subtree of T_k, on the path from u to $\mathsf{root}(T_k)$, such that u does *not* belong to it, u must store a number of keys proportional to the number of nodes in the sub-subtree. That is

$$\sum_{k=1}^{\log n} (2^k - k) \approx O(n)$$

We would like to reduce this to $O(\log n)$ per sub-subtree. We do that by using pseudorandom generators.

Let G be a PRG that triples its input

$$G(s) = G_L(s) \circ G_M(s) \circ G_R(s)$$

First of all we give random labels to each node which is not a leaf. Let LAB_i the label of node v_i. Now consider the subtree T_i rooted at v_i. We apply a top-down labeling procedure which assigns more labels to each node. The label of the left (resp. right) child is G_L (resp. G_R) applied to LAB of the parent.

Now a node v_j has $\log n$ labels. If v_i is an ancestor of v_j, with LAB_{ij} we denote the label assigned to v_j by the labeling procedure started at v_i. The subset S_{ij} is assigned the key $L_{ij} = G_M(LAB_{ij})$.

This labeling system has the following properties:

- given LAB_i it is possible to compute the keys of all the subsets S_{ij};
- without knowing LAB_i (or any intermediate label in the labeling procedure started at v_i) the keys of all subsets S_{ij} are pseudorandom.

So if u is a user, what does it get? For each subtree T_i in which u is a leaf, consider the path from the $\mathsf{root}(T_i)$ to u. Give to u all the labels of the roots of the "hanging" subtrees. Thus u gets up to $O(\log n)$ labels per subtree which makes $O(\log^2 n)$ total.

How do we decrypt? The cover can be found in $O(\log \log n)$ steps. User u can compute the right key from the labels in its possession, via $O(\log n)$ applications of the PRG G. Once the key is found a single decryption is required.

5.3 Bibliographical Note

The problem of broadcast encryption was introduced in [8], where a combinatorial solution was presented. Their solution is similar in spirit to the MAC-only multicast authentication algorithm presented in Section 4.2. In fact the work in [8] predates and inspired the MAC-only algorithm.

The simple tree scheme in Section 5.1 was independently proposed in [19] and [20]. The stateless receiver solution was presented in [14].

6 Tracing Traitors

The previous broadcast encryption schemes, guarantee that only the correct subset of users is allowed to decrypt the data. However the system can be easily circumvented by a single authorized user who reveal its private information to allow other non-authorized users to decrypt.

Consider the example of an encrypted Pay-TV station. A legitimate receiver can do two things:

- re-broadcast the clear data that it receives. This is however, costly, complicated and easily detectable.
- copy the secret key in its possession and "sell" a "pirate" decoder box on the black market.

The second attack is much harder to detect. It is usually countered by providing users with tamper-resistant decoders, which prevents direct reading or copying of the internal keys. However, very secure tamper-proof hardware is extremely costly, while basic solutions can easily be reversed-engineer.

Thus we are interested in trying to devise a software-only solution to help against the "traitors" problem.

The basic idea is to somehow "fingerprint" the keys of each user. When these keys are found inside a decoder box, then we can trace them back to the "traitor user".

It should be possible to trace at least one such traitor, even if the pirate decoder box was built using the information from a coalition of up to k traitors (k being a parameter in the scheme).

Moreover, we are not sure if when we find a pirate decoder box, we will be able to open it and read its content (the pirate box might be tamper-resistant as well). Thus we would like the tracing mechanism to be *black-box*, i.e. it should identify the traitor by simple I/O interaction with the pirate box.

In this section we will discuss some "tracing traitors" algorithms. Such a scheme should be implemented on top of any of the broadcast encryption schemes presented before. A general way to do this is to split the message being sent as

$M = M_1 \oplus M_2$ and then encrypt M_1 with the broadcast encryption scheme, and M_2 with the traitor tracing scheme.

The parameters of a traitor tracing scheme are n, the number of users, and k, the maximum size of a coalition of traitors that the scheme tolerates.

6.1 Tracing Traitors in the Subset Cover Framework

The Subset Cover framework for broadcast encryption (discussed in Section 5.2) provides a seamless integration of broadcast encryption with traitor tracing.

We are going to present a traitor tracing mechanism that works for *any* instantiation of the subset cover mechanism that satisfies the so-called *subset bifurcation* property.

Recall that in the subset cover framework we partition the privileged set $N - R$ into m disjoint subsets $S_{i_1}, \ldots, S_{i_m}$ from a predetermined family. We say that the *subset bifurcation* property is satisfied if there exists a constants a_1, a_2 such that, for all S_i in the family, there exists Si_L, S_{i_R} also in the family, such that: (1) $S_{i_L} \cup S_{i_R} = S_i$, S_{i_L} and S_{i_R} are disjoint, $a_1|S_{i_L}| < |S_i| < a_2|S_{i_L}|$.

Before we show how to use the subset bifurcation property to perform traitor tracing, we show that the two instantiations of the subset cover framework shown in Section 5.2 have this property.

The Complete Subtree Method. Each subset is formed by the leaves of a complete subtree. Each subtree can be split as the left and the right subtree. Thus the constant $a = 1/2$.

The Subset Difference Method. Each subset S_{ij} can be split as the left and tight subtree of i. The worst case is when i is the grandparent of j, in which case one side gets 2/3 of the elements, while the other only 1/3.

The Tracing Mechanism. We use the bifurcation property to perform a sort of binary search on the users. We are given a pirate box which decodes with a certain cover, i.e. decrypts the ciphertext[2]

$$C =< i_1, \ldots, i_m, E_{L_1}(K), \ldots, E_{L_m}(K), E'_K(M) >$$

Suppose we have a way to identify a subset S_{i_j} in the cover, such that a traitor belongs to S_{i_j} (clearly at least one traitor must belong to a least one subset).

If $|S_{i_j}| = 1$ then we are done. Otherwise we split S_{i_j} using the bifurcation property and continue on.

Thus all we need to show is how to identify such a subset S_{i_j}. Recall that we may have many traitors, thus the box may have several keys in it. Recall also that we can only query the box to get decrypted values (black-box access).

We look at the behavior of the box when it is fed ciphertexts of the following form:

$$< i_1, \ldots, i_m, E_{L_1}(R_K), \ldots, E_{L_j}(R_K), E_{L_{j+1}}(K), \ldots, E_{L_m}(K), E'_K(M) >$$

[2] We are going to assume that the pirate box decrypts with probability 1 given a certain cover. This is not necessary, since we can adjust for a "threshold" probability of decryption p.

We denote with p_j the probability that the box decrypts correctly this "faulty" ciphertext. Basically if p_j is sufficiently high, we know that the box must contain L_{i_k} for some $k > j$ and thus the traitor is in one of the subsets S_{i_k}, for $k > j$.

Notice that $p_0 = 1$, while $p_m = 0$, thus we must have that for some j

$$|p_{j-1} - p_j| \geq \frac{1}{m}$$

If p_j is substantially different from p_{j-1} it means that the box is using the correct key L_{i_j} and thus S_{i_j} must contain a traitor.

Once again we find the relevant p_j by a sort of binary search using the following procedure:

$ST(0, m)$
1. $a \leftarrow 0$;
2. $b \leftarrow 0$;
3. $c \leftarrow \frac{a+b}{2}$;
4. **Estimate** p_a, p_b, p_c;
5. **If** $|p_c - p_a| \geq |p_b - p_c|$;
6. **Then** $ST(a, c)$;
7. **Else** $ST(c, b)$;

6.2 A Public-Key Scheme

We conclude with a public-key scheme. It is mostly of theoretical interest since in this scenario, private-key encryption is usually adopted for efficiency reasons. However it is interesting because it shows that algebraic techniques, and not just combinatorial ones, can be used for these applications. Also being a public-key scheme it allows *anybody* to broadcast messages to the receivers, and not just a pre-determined sender.

ElGamal Encryption. Let G_q be a group of prime order q. Let g be a generator for G_q. The public key of the ElGamal encryption scheme is a value $y = g^x$ with $x \in_R Z_q$, which is the secret key. A message $m \in G_q$ is encrypted by a pair (α, β) where $\alpha = g^r$ and $\beta = y^r \cdot m$ with $r \in_R Z_q$. The decryption step consists of retrieving $m = \beta \cdot \alpha^{-x}$.

The traitor tracing scheme. The new scheme is a generalization of the ElGamal scheme. The public key is modified as follows. For each $i = 1, \ldots, 2k$ we choose $r_i, \alpha_i \in_R Z_q$ and set $h_i = g^{r_i}$, $y = \prod_{i=1}^{2k} h_i^{\alpha_i}$. The public key is $< y, h_1, h_2, \ldots, h_{2k} >$. Notice that given the public key, there are several possible combinations of the α's (exactly q^{2k-1}) that yield $y = \prod_{i=1}^{2k} h_i^{\alpha_i}$. Such a vector $< \alpha_1, \ldots, \alpha_{2k} >$ is called a *representation* of y with respect to $< h_1, \ldots, h_{2k} >$.

Finally we also assume that there is a publicly known code Γ, which is a collection of n codewords in Z^{2k}.

$$\Gamma = \{\gamma^{(1)}, \ldots, \gamma^{(n)}\} \quad \text{with} \quad \gamma^{(i)} =< \gamma_1^{(i)}, \ldots, \gamma_{2k}^{(i)} >$$

User u_i gets a value $\theta_i \in Z_q$ such that $\theta_i \cdot \gamma^{(i)}$ is a representation of y:

$$\theta_i = \frac{\sum r_j \alpha_j}{\sum r_j \gamma_j^{(i)}} \bmod q$$

We denote with $d^{(i)} = \theta_i \cdot \gamma^{(i)}$ the representation held by u_i.

To encrypt $m \in G_q$ we generalize ElGamal as follows:

$$E_r(m) =< h_1^r, h_2^r, \ldots, h_{2k}^r, y^r \cdot m >$$

To decrypt a ciphertext $c =< H_1, \ldots, H_{2k}, S >$ the user u_i computes $m = S \cdot U^{-\theta_i}$ where $U = \prod_{J=1}^{2k} H_j \gamma_j^i$. Notice that this is a good decryption given any representation.

The code Γ is what enables the tracing mechanism. Indeed it is easy to show that if an adversary gets $d^{(1)}, \ldots, d^{(\ell)}$, the representations of ℓ users, where $\ell < k$, then she can only compute new representations d which are *convex* combinations of the $d^{(i)}$'s i.e.

$$d = \sum_{i=1}^{\ell} a_i d^{(i)} \quad \text{where} \quad \sum_{i=1}^{\ell} a_i = 1$$

If we instantiate the code Γ as the Reed-Solomon code, then it is possible, given d, to identify *all* the components $d^{(i)}$ used in the linear combination.

Notice that the above approach requires d and as such is not a *black-box* tracing mechanism. The paper also presents a black-box mechanism, but it is inefficient since it runs in time $O(\binom{n}{k})$ so it is feasible only for small coalitions.

6.3 Bibliographical Note

The problem of tracing the source of pirate decoders was introduced in [6], which appeared in its final journal form in [7]. The tracing mechanism for stateless receivers was proposed in [14] The public-key scheme originates from [3].

References

1. R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas and R. Needham. A New Family of Authentication Protocols. *Operating Systems Review*, 32(4);9–20. October 1998.
2. F. Bergadano, D. Cavagnino and B. Crispo. Individual Source Authentication on the MBONE. *Proceedings of ICME 2000.*
3. D. Boneh and M .Franklin. An Efficient Public-Key Traitor Tracing Scheme. *Proceedings of CRYPTO'99*, LNCS vol.1666 pp.338–353. Springer 1999.
4. R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor and B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. *Proceedings of INFOCOM'99.* 1999.
5. S. Cheung. An Efficient Message Authentication Scheme for Link State Routing. *In Proceedings of the 13th Annual Computer Security Applications Conference.* 1997.
6. B. Chor, A. Fiat and M. Naor. Tracing Traitors. *Proceedings of CRYPTO'94*, LNCS vol.839, pp.257–270. Springer 1994.
7. B. Chor, A. Fiat, M. Naor and B. Pinkas. Tracing Traitors. *IEEE Transactions on Information Theory*, 46:3, May 200.

8. A. Fiat and M. Naor. Broadcast Encryption. *Proceedings of CRYPTO'93*. LNCS vol.773, pp.480–491. Springer 1994.
9. R. Gennaro and P. Rohatgi. How to Sign Digital Streams. *Information and Computation*, 165:100–116. 2001.
10. S. Goldwasser, S. Micali and R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen Message Attack. *SIAM J. Comp.* 17(2):281–308, 1988.
11. L. Lamport. Constructing Digital Signatures from a One-Way Function. *Technical Report SRI Intl.* CSL 98, 1979.
12. R. Merkle. A Digital Signature based on a Conventional Encryption Function. *Advances in Cryptology–Crypto'87*. LNCS, vol.293, pp. 369–378, Springer–Verlag, 1988.
13. R. Merkle. A Certified Digital Signature. *Advances in Cryptology–Crypto'89*. LNCS, vol.435, pp. 218–238, Springer–Verlag, 1990.
14. D. Naor, M. Naor and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. *Proceedings of CRYPTO 2001*, LNCS vol.2139, pp.41–62. Springer 2001.
15. National Institute of Standard and Technology. Secure Hash Standard. NIST FIPS Pub 180-1, 1995.
16. A. Perrig, R. Canetti, J. Tygar and D. Song. Efficient Authentication and Signature of Multicast Streams over Lossy Channels. *Proceedings of the 2000 IEEE Symposium on Security and Privacy*.
17. P. Rohatgi. A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication. *Proceedings of ACM CCS '99*, pp. 93–100.
18. D. Stinson. *Cryptography. Theory and Practice.* CRC Press. 1996.
19. D. Wallner, E. Harder and R. Agee. Key Management for Multicast: Issues and Architectures. IETF Draft wallner-key, July 1997. Available from ftp://ftp.ietf.org/internet-drafts/draft-wallner-key-arch-01.txt
20. C. Wong, M. Gouda and S. Lam. Secure Group Communications Using Key Graphs. *SIGCOMM 1998*.

Security for Mobility

Hanne Riis Nielson, Flemming Nielson, and Mikael Buchholtz

Informatics and Mathematical Modelling, Technical University of Denmark
Richard Petersens Plads, Building 321, DK-2800 Lyngby, Denmark
{riis,nielson,mib}@imm.dtu.dk

Abstract. We show how to use static analysis to provide information about security issues related to mobility. First the syntax and semantics of Mobile Ambients is reviewed and we show how to obtain a so-called 0CFA analysis that can be implemented in polynomial time. Next we consider discretionary access control where we devise Discretionary Ambients, based on Safe Ambients, and we adapt the semantics and 0CFA analysis; to strengthen the analysis we incorporate context-sensitivity to obtain a 1CFA analysis. This paves the way for dealing with mandatory access control where we express both a Bell-LaPadula model for confidentiality as well as a Biba model for integrity. Finally, we use Boxed Ambients as a means for expressing cryptographic key exchange protocols and we adapt the operational semantics and the 0CFA analysis.

1 Introduction

Mobile Ambients (see Section 2) were introduced by Cardelli and Gordon [13,16] as a formalism for reasoning about mobility. Ambients present a high-level view of mobile computation and give rise to a high-level treatment of the related security issues.

An ambient is a named bounded place and the boundary determines what is outside and what is inside. Ambients can be nested inside each other and thereby form a tree structure. Mobility is then represented as navigation inside this hierarchy. Each ambient contains a number of multi-threaded running processes; the top-level processes of each ambient have direct control over it and can instruct it to move and thereby change its future behaviour. The ambient names are unforgeable and are essential for controlling access to the ambients. As in [12] we shall impose a simple type structure by assigning groups to ambients.

The basic calculus has three so-called subjective mobility capabilities: an enclosing ambient can be instructed to move into a sibling ambient, it can be instructed to move out of its enclosing ambient, and a sibling ambient can be dissolved. The literature contains a number of extensions to the basic calculus: so-called objective moves, various forms of communication and primitives for access control etc; we shall begin by considering the basic calculus in Section 2, then add access control features in Sections 3 and 4, and finally revert to the basic calculus in order to add communication primitives in Section 5.

The operational semantics is a standard reduction semantics with a structural congruence relation. The static analysis is modelled on the simple 0CFA analysis

R. Focardi and R. Gorrieri (Eds.): FOSAD 2001/2002, LNCS 2946, pp. 207–265, 2004.

originally developed for functional programs. In the case of Mobile Ambients the control structure is expressed by the hierarchical structure of ambients (with separate components taking care of the communication, if present). Hence we aim at modelling the *father-son* relationship between the nodes of the tree structure [31,30].

The precision of the 0CFA analysis is roughly comparable to that of early type systems for Mobile Ambients [11,14] and may be used for validating security properties related to *crossing control* and *opening control* [12]. In the spirit of type systems the main semantic result showing the correctness of the 0CFA analysis is a subject-reduction result expressing that the analysis information remains valid during execution.

The efficiency of the analysis is good and the worst-case complexity is cubic [36]. In practical terms we find it convenient to translate the analysis into a fragment of first order logic known as Alternation-free Least Fixed Point Logic (ALFP) and implemented by our Succinct Solver [35].

Discretionary access control (see Section 3) imposes conditions on when an ambient can perform a given mobility primitive on another ambient. As an example, an ambient (the *subject*) may move into another ambient (the *object*) by executing a suitable capability (an *access operation*). In the *Safe Ambients* of Levi and Sangiorgi [24] there is a simple notion of access control; here the object must agree to being entered and this is expressed by requiring the object to execute the corresponding co-capability (an *access right*).

This rudimentary kind of access control does not fully model the usual notion of access control where an *access control matrix* lists the set of capabilities that each subject may perform on each object. (In the classical setting [22], the subjects correspond to programs, the objects correspond to files, and the access operations could be the read, write, and execute permissions of UNIX.) We overcome this shortcoming by designing the *Discretionary Ambients* where co-capabilities not only indicate the access rights but also the subject that is allowed to perform it.

We then adapt the semantics to incorporate the necessary checks and hence to block execution whenever an inadmissible access operation is performed. Similarly we adapt the analysis and later strengthen it using context-sensitivity; this is a standard technique from data flow and control flow analysis that can be used to improve the precision of a simple 0CFA analysis in order to obtain a so-called 1CFA analysis [29]. As mentioned above the 0CFA analysis approximates the hierarchical structure of the ambients by a binary *father-son* relationship. Context-sensitivity then is based on the observation that more precise results are likely to be obtained using a ternary *grandfather-father-son* relationship between ambients. This 1CFA analysis still has reasonable complexity and we report on practical experiments confirming that the use of ternary relations strikes a useful balance between precision and efficiency. (A considerably more precise and costly analysis is presented in [37,38].)

Mandatory access control (see Section 4) imposes confidentiality and/or integrity by combining the access control matrices with additional information [22]. The Bell-LaPadula model assigns security levels to objects and subjects and imposes *confidentiality* by preventing information from flowing downwards from a high security level to a low security level. The Biba model assigns integrity levels to objects and subjects and then imposes *integrity* by preventing trusted high-level entities from being corrupted by dubious low-level entities — thus information is prevented from flowing upwards.

These security models were originally developed in a setting much more static than the one of Discretionary Ambients. For comparison, an ambient may be viewed as a system with a distributed access control matrix that dynamically evolves and that is concerned with multiplicities whereas classically the access control matrix is partly centralised and static. In this paper we show how the security policies may be re-formulated in the dynamic and mobile setting of the Discretionary Ambients.

The formal development amounts to adapting the semantics so as to incorporate reference monitors that block execution whenever an inadmissible access operation is performed (according to the mandatory access control policy considered). The analysis is extended to perform tests comparable to those of the reference monitors, and as an extension of the subject reduction theorem we show that if all static tests are satisfied then the reference monitor can be dispensed with.

Cryptographic protocols (see Section 5) are most naturally expressed using communication. The full calculus of Mobile Ambients includes a notion of local communication where there is a communication box inside each ambient; this naturally leads to dealing with asynchronous communication. For some purposes it is more convenient to allow communication between adjacent layers of ambients and this motivated the design of Boxed Ambients [9,8]. Here an ambient can directly access not only its local communication box but also the communication box of its parent (but not grandparents) as well as its children (but not grandchildren). We show that perfect symmetric cryptography as well as a number of cryptographic key exchange protocols (Wide Mouthed Frog, Yahalom and the Needham-Schroeder symmetric key protocol) can be expressed in a rather natural manner in Boxed Ambients. We adapt the semantics and the 0CFA analysis to this setting and prove the usual results; thanks to a small simplification also the implementation is relatively straightforward. This analysis may additionally be used for validating security properties related to *exchange analysis* as presented in [12].

In the *Conclusion* (see Section 6) we summarise the development performed and briefly discuss extensions of the work as well as directions for future research: the notion of hardest attackers as a means for characterising all Dolev-Yao attackers to a firewall [31,30], and the possibility of extending this to capture all Dolev-Yao attackers to the protocols considered [5].

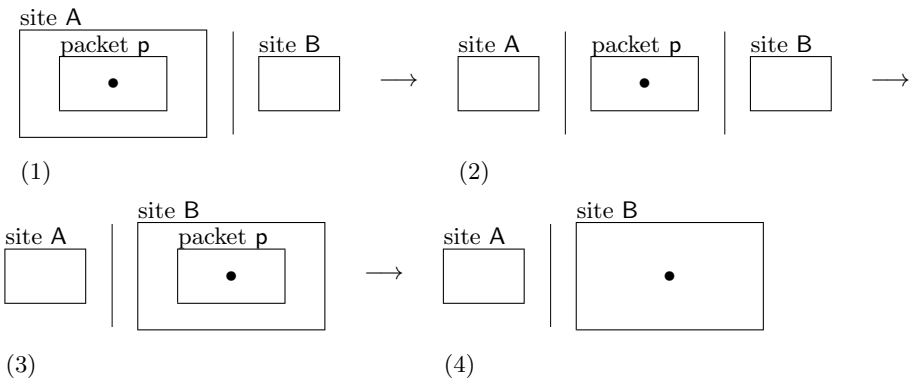


Fig. 1. A packet p moves from site A to site B and finally gets dissolved.

2 Mobile Ambients

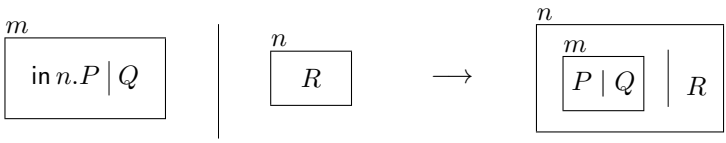
In the ambient view of the world each ambient is a bounded place where computations take place. The boundary determines what is inside and what is outside and as such it represents is a high-level security abstraction. Additionally it provides a powerful abstraction of mobility where ambients move as a whole. This view is sufficiently flexible to apply to a variety of scenarios: applets, agents, laptops, etc.

Ambients can be nested inside other ambients forming a tree structure. Mobility is then represented as navigation within this hierarchy of ambients. As an example, consider Figure 1 where a packet p moves from one site A into another site B. First we move the packet out the enclosing ambient (2) and then into the new enclosing ambient (3). Finally in (4), the payload of the packet is opened inside site B and the packet p is, thereby, dissolved.

Each ambient contains a number of multi-threaded running processes. The top-level processes of an ambient have direct control over it and can instruct it to move and thereby change the future behaviour of its processes and sub-ambients; consequently, the processes of sub-ambients only control the sub-ambient in which they are placed. Processes continue to run while being moved.

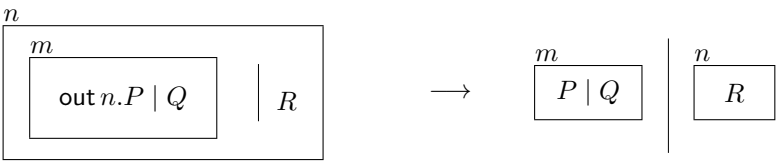
Each ambient has a name. Only the name can be used to control the access to the ambient (entry, exit, etc.) and the ambient names are unforgeable.

The mobility primitives of ambients are based on the notion of subjective moves. Here the movements of an ambient are caused by the threads running at top-level inside it. The in-capability directs an ambient to move into a sibling (i.e. another ambient running in parallel). This can be depicted as:



The left hand side shows two sibling ambients named m and n; the ambient m has a thread instructing it to move into the ambient n. The right hand side of the figure then shows the result of this action.

The out-capability directs an ambient to move out of its father (i.e. the enclosing ambient). In the figure below the ambient m contains a thread instructing it to move out of its father named n:



The open-capability allows a process to dissolve the boundary around a sibling ambient (named n below):

The ambient view of systems directly focuses on the ability to express a number of high-level security issues related to mobility; as for example, ensuring that packets with sensitive information can only pass through classified sites, or that packets with sensitive information may pass through unclassified sites but can only be opened at classified sites.

2.1 Syntax and Semantics of Mobile Ambients

To make this precise we formally define the syntax of processes, P, and capabilities, M, by the following grammar:

Processes based on the π-calculus:

$P ::= (\nu\, n : \mu)\, P$	introduces a process with private name n in group μ	
$\mid\ (\nu\, \mu)\, P$	introduces a new group named μ with its scope	
$\mid\ \mathbf{0}$	the inactive process	
$\mid\ P_1 \mid P_2$	two concurrent processes	
$\mid\ !P$	replication: any number of occurrences of P	
$\mid\ n\,[P]$	an ambient named n containing P	(drawn as $\overset{n}{\boxed{P}}$)
$\mid\ M.P$	a capability M followed by P	

Capabilities of the core calculus:

$M ::= \mathsf{in}\, n$	move the enclosing ambient into a sibling named n
$\mid\ \mathsf{out}\, n$	move the enclosing ambient out of a parent named n
$\mid\ \mathsf{open}\, n$	dissolve a sibling ambient named n

In the graphical representation the inactive process is usually not written explicitly. Our syntax of ambients follows [12] and extends the basic calculus of [13,16] in not having an operation $(\nu\, n)P$ for introducing a new private name

Table 1. The structural congruence relation.

$P \equiv P$	$!P \equiv P \mid !P$	
$P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$	$!\mathbf{0} \equiv \mathbf{0}$	
$P \equiv Q \Rightarrow Q \equiv P$	$(\nu\, n\colon \mu)\, \mathbf{0} \equiv \mathbf{0}$	
	$(\nu\, \mu)\, \mathbf{0} \equiv \mathbf{0}$	
$P \equiv Q \Rightarrow (\nu\, n\colon \mu)\, P \equiv (\nu\, n\colon \mu)\, Q$		
$P \equiv Q \Rightarrow (\nu\, \mu)\, P \equiv (\nu\, \mu)\, Q$	$(\nu\, n\colon \mu)\, (\nu\, n'\colon \mu')\, P \equiv$	
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	$\qquad (\nu\, n'\colon \mu')\, (\nu\, n\colon \mu)\, P$	if $n \neq n'$
$P \equiv Q \Rightarrow\ !P \equiv\ !Q$	$(\nu\, \mu)\, (\nu\, \mu')\, P \equiv (\nu\, \mu')\, (\nu\, \mu)\, P$	
$P \equiv Q \Rightarrow n\,[P] \equiv n\,[Q]$	$(\nu\, n\colon \mu)\, (\nu\, \mu')\, P \equiv (\nu\, \mu')\, (\nu\, n\colon \mu)\, P$	if $\mu \neq \mu'$
$P \equiv Q \Rightarrow M.\, P \equiv M.\, Q$		
	$(\nu\, n\colon \mu)\, (P \mid Q) \equiv P \mid (\nu\, n\colon \mu)\, Q$	if $n \notin \mathrm{fn}(P)$
$P \mid Q \equiv Q \mid P$	$(\nu\, \mu)\, (P \mid Q) \equiv P \mid (\nu\, \mu)\, Q$	if $\mu \notin \mathrm{fg}(P)$
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$		
$P \mid \mathbf{0} \equiv P$	$(\nu\, n'\colon \mu)\, (n\,[P]) \equiv n\,[(\nu\, n'\colon \mu)\, P]$	if $n \neq n'$
	$(\nu\, \mu)\, (n\,[P]) \equiv n\,[(\nu\, \mu)\, P]$	
	$(\nu\, n\colon \mu)\, P \equiv (\nu\, n'\colon \mu)\, (P\{n \leftarrow n'\})$	if $n' \notin \mathrm{fn}(P)$

Table 2. The transition relation for Mobile Ambients.

$$\frac{P \to Q}{(\nu\, n\colon \mu)\, P \to (\nu\, n\colon \mu)\, Q} \qquad \frac{P \to Q}{(\nu\, \mu)\, P \to (\nu\, \mu)\, Q}$$

$$\frac{P \to Q}{n\,[P] \to n\,[Q]} \qquad \frac{P \to Q}{P \mid R \to Q \mid R} \qquad \frac{P \equiv P' \wedge P' \to Q' \wedge Q' \equiv Q}{P \to Q}$$

$$m\,[\mathsf{in}\, n.\, P \mid Q] \mid n\,[R] \ \to\ n\,[m\,[P \mid Q] \mid R]$$

$$n\,[m\,[\mathsf{out}\, n.\, P \mid Q] \mid R] \ \to\ m\,[P \mid Q] \mid n\,[R]$$

$$\mathsf{open}\, n.\, P \mid n\,[Q] \ \to\ P \mid Q$$

n for use in P but instead two operations: $(\nu\, \mu)\, P$ for introducing a new group name μ for use in P and then $(\nu\, n\colon \mu)\, P$ for introducing a new private name n belonging to the group μ. A group can be viewed as the "type" of a name and has no semantic consequence.

The importance of groups becomes clear in the 0CFA analysis where we will need that the group name is stable under α-renaming of names. We achieve this by means of a careful definition of the structural congruence (see Table 1 and the explanation below). For simplicity there will be no α-renaming of group names; for this to work we make the simplifying assumption that all $(\nu\, \mu)$ must be distinct and must not occur inside some replication operator (!).

The semantics of mobile ambients consists of a structural congruence relation, written $P \equiv Q$, and a transition relation, written $P \to Q$. The structural congruence relation allows to rearrange the syntactic appearance of processes

as the pictorial representation suggests (e.g. $P \mid Q \equiv Q \mid P$), it deals with the semantics of replication (e.g. $!P \equiv !P \mid P$) and allows to perform α-renaming; the details are fairly standard and may be found in Table 1. We write $\mathrm{fn}(P)$ and $\mathrm{fg}(P)$ for the free names and the free groups of P, respectively. The transition relation formalises the three subjective moves and clarifies that capabilities deeply nested inside ambients may execute provide they are not prefixed with other capabilities; the details are fairly standard and may be found in Table 2.

Example 1. Let us consider a packet p moving from a site A to another site B as in Figure 1. The first configuration (1) in Figure 1 could be the process:

$$\mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}]] \mid \mathsf{B}\,[\mathsf{open\ p}]$$

Using the transition relation of Table 2 we can get an execution sequence corresponding to that of Figure 1:

$$\begin{array}{lll} (1) & \mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}]] \mid \mathsf{B}\,[\mathsf{open\ p}] & \rightarrow \\ (2) & \mathsf{A}\,[\,] \mid \mathsf{p}\,[\mathsf{in\ B}] \mid \mathsf{B}\,[\mathsf{open\ p}] & \rightarrow \\ (3) & \mathsf{A}\,[\,] \mid \mathsf{B}\,[\mathsf{open\ p} \mid \mathsf{p}\,[\,]] & \rightarrow \\ (4) & \mathsf{A}\,[\,] \mid \mathsf{B}\,[\,] & \end{array}$$

□

2.2 A 0CFA Analysis for Mobile Ambients

The aim of the static analysis is to determine which ambients and capabilities may turn up inside given ambients. Fixing our attention on a given ambient process P our aim is to find an estimate $\mathcal{I}$ of this information that describes all configurations reachable from P. In the Flow Logic approach to static analysis [28,39] we proceed in the following stages:

Specification: First we define what it means for the estimate $\mathcal{I}$ to be an acceptable description of the process P.

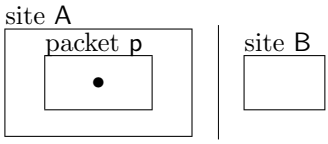
Correctness: Then we prove that all acceptable estimates will remain acceptable during execution.

Implementation: Finally, we show that a best acceptable estimate can be calculated in polynomial time.

This approach should be rather natural to readers familiar with type systems: first one formulates the type system (thereby making precise the notion of *type checking*), then one shows semantic soundness of the type system (usually in the form of a *subject-reduction* result), and finally one shows how to obtain principal types for processes (thereby making precise the notion of *type inference*).

Example 2. Consider the Mobile Ambient process of Example 1

$$\mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}]] \mid \mathsf{B}\,[\mathsf{open\ p}]$$



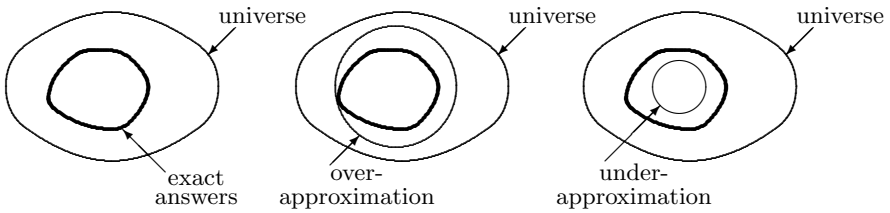


Fig. 2. The nature of approximation.

where ambients A and B belong to the group $\mathbb{S}$ of sites, and the ambient p belongs to the group $\mathbb{P}$ of packets. The analysis will provide a safe approximation of which ambients may turn up inside other ambients.

The exact answer is that p may turn up inside A, p may turn up inside B, but that A and B never turn up inside p nor inside each other. In terms of groups we shall simply say that $\mathbb{P}$ may turn up inside $\mathbb{S}$ but that $\mathbb{S}$ never turns up inside $\mathbb{P}$ nor inside $\mathbb{S}$. We shall represent this using a mapping

$$\mathcal{I} : \mathbf{Group} \to \mathcal{P}(\mathbf{Group} \cup \mathbf{Cap})$$

that for each ambient group $\mu \in \mathbf{Group}$ tells us not only which ambient groups may be inside an ambient in group μ but also which group capabilities may be possessed by an ambient in group μ; here a group capability $m \in \mathbf{Cap}$ is given by:

$$m ::= \mathsf{in}\,\mu \mid \mathsf{out}\,\mu \mid \mathsf{open}\,\mu$$

The optimum value of $\mathcal{I}$ for the example discussed above is given by

$$\begin{array}{lcl} \mathcal{I}(\star) & = & \{\mathbb{S}, \mathbb{P}\} \\ \mathcal{I}(\mathbb{S}) & = & \{\mathbb{P}, \mathsf{open}\,\mathbb{P}\} \\ \mathcal{I}(\mathbb{P}) & = & \{\mathsf{in}\,\mathbb{S}, \mathsf{out}\,\mathbb{S}\} \end{array}$$

where $\star$ denotes the group of the overall system (i.e. the top level). A somewhat less precise *over-approximation* of $\mathcal{I}$, where extra elements are included, is

$$\begin{array}{lcl} \mathcal{I}(\star) & = & \{\mathbb{S}, \mathbb{P}\} \\ \mathcal{I}(\mathbb{S}) & = & \{\mathbb{P}, \mathbb{S}, \mathsf{in}\,\mathbb{S}, \mathsf{out}\,\mathbb{S}, \mathsf{open}\,\mathbb{P}\} \\ \mathcal{I}(\mathbb{P}) & = & \{\mathsf{in}\,\mathbb{S}, \mathsf{out}\,\mathbb{S}\} \end{array}$$

and it will turn out that this is the one that will be obtained using the simplest of our analyses (the 0CFA analysis developed in this subsection). □

Remark 3. The choice of the domain of $\mathcal{I}$ determines which kind of information the analysis can provide about a process and, consequently, what it *cannot* record.

For example, with the choice of $\mathcal{I}$ as in Example 2 we can record whether an ambient or a capability is present inside some other ambient, but not *the number* of ambients and capabilities that are present. To get such information we will have to change the domain of the analysis estimate as done in e.g. [37,23,38].

Furthermore, we can only records the presence of capabilities but not the order in which they appear. Thus, we do not capture the order in which capabilities are executed and cannot determine whether one capability is executed before another; in other words the analysis is not flow-sensitive. Adding sequences of capabilities have been studied in [23].

In the remainder of this paper we *do not* consider neither multiplicities nor flow-sensitivity. As we will see, even these "simple" analyses are able to give analysis estimates that are sufficiently precise to determine interesting security properties. □

Specification of the 0CFA Analysis. In the above example we displayed the best value of $\mathcal{I}$ that one can hope for. In general this is not always possible since the problem of finding the best value of $\mathcal{I}$ is really undecidable due to the Turing completeness of Mobile Ambients. Hence we will have to settle for more approximate estimates saying that $\mathbb{S}$ may turn up inside $\mathbb{P}$, whereas we shall never accept an estimate saying that $\mathbb{P}$ never turns up inside $\mathbb{S}$. In terms of approximation this means that we opt for an over-approximation of the set of containments; this is illustrated in Figure 2.

To make precise when an estimate $\mathcal{I} : \mathbf{Group} \to \mathcal{P}(\mathbf{Group} \cup \mathbf{Cap})$ describes an acceptable over-approximation of the behaviour of a process P under consideration we shall axiomatise a judgement

$$\mathcal{I} \models^{\mu}_{\Gamma} P$$

meaning that $\mathcal{I}$ is an acceptable analysis estimate for the process P when it occurs inside an ambient from the group μ and whenever the ambients are in groups as specified by the type environment Γ (e.g. $\Gamma(\mathsf{p}) = \mathbb{P}$ and $\Gamma(\mathsf{A}) = \Gamma(\mathsf{B}) = \mathbb{S}$). The judgement is defined by structural induction on the syntax of the process P (as shown below).

Analysis of Composite Processes. Each acceptable analysis estimate for a composite process must also be an acceptable analysis estimate for its sub-processes; perhaps more imprecise than need be. This is captured by the following clauses where Γ is the current type environment and $\star$ is the ambience i.e. the group associated with the enclosing ambient.

$\mathcal{I} \models^{\star}_{\Gamma} (\nu\, n\colon \mu)\, P$	iff $\mathcal{I} \models^{\star}_{\Gamma[n \mapsto \mu]} P$		update type env.; check process
$\mathcal{I} \models^{\star}_{\Gamma} (\nu\, \mu)\, P$	iff $\mathcal{I} \models^{\star}_{\Gamma[\mu \mapsto \diamond]} P$		update type env.; check process
$\mathcal{I} \models^{\star}_{\Gamma} \mathbf{0}$	iff true		nothing to check
$\mathcal{I} \models^{\star}_{\Gamma} P_1 \mid P_2$	iff $\mathcal{I} \models^{\star}_{\Gamma} P_1 \ \wedge\ \mathcal{I} \models^{\star}_{\Gamma} P_2$		check both branches
$\mathcal{I} \models^{\star}_{\Gamma} !P$	iff $\mathcal{I} \models^{\star}_{\Gamma} P$		check process; ignore multiplicity
$\mathcal{I} \models^{\star}_{\Gamma} n\,[P]$	iff $\mu \in \mathcal{I}(\star) \ \wedge\ \mathcal{I} \models^{\mu}_{\Gamma} P$ where $\mu = \Gamma(n)$		μ is inside $\star$; check process

In the first clause we update the type environment with the type of the newly introduced name; in the second clause we update the type environment with a special placeholder $\diamond$ indicating a group name; in the last clause we ensure that the analysis estimate $\mathcal{I}$ records that the group of n occurs inside the ambience $\star$ and we analyse the internals of n in an appropriately updated ambience.

Remark 4. Elaborating on the analogy to type systems explained above, one could coin the slogan:

> Flow Logic is the approach to static analysis that presents Data and Control Flow Analysis as a Type System.

To make this more apparent we could formulate the first clause above as an inference rule

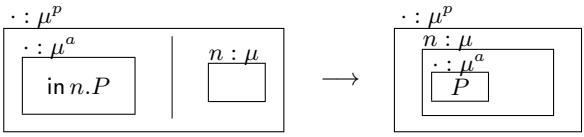
$$\frac{\mathcal{I} \models^{\star}_{\Gamma[n \mapsto \mu]} P}{\mathcal{I} \models^{\star}_{\Gamma} (\nu\, n\colon \mu)\, P}$$

and similarly for the other clauses presented here. These formulations are equivalent[1] whenever the judgement is defined by structural induction on processes. The formulation chosen here is perhaps more in the spirit of the equational approach of Data Flow Analysis. □

In-*Capability.* Each acceptable analysis estimate must mimic the semantics: if the semantics allows one configuration to evolve into another then it must be reflected in the analysis estimate. For the in-capability this is achieved by the following clause:

$$\begin{array}{lll}
\mathcal{I} \models^{\star}_{\Gamma} \mathsf{in}\, n.\ P \text{ iff } & \mathsf{in}\ \mu \in \mathcal{I}(\star)\ \wedge\ \mathcal{I} \models^{\star}_{\Gamma} P\ \wedge & \\
& \forall\ \mu^a, \mu^p : \mathsf{in}\ \mu \in \mathcal{I}(\mu^a)\ \wedge & \mu^a \text{ has the capability } \mathsf{in}\,\mu \\
& \quad \mu^a \in \mathcal{I}(\mu^p)\ \wedge & \mu^p \text{ is the parent of } \mu^a \\
& \quad \mu \in \mathcal{I}(\mu^p) & \mu^a \text{ has a sibling } \mu \\
& \Rightarrow\ \mu^a \in \mathcal{I}(\mu) & \mu^a \text{ may move into } \mu \\
& \text{where } \mu = \Gamma(n) &
\end{array}$$

Here the first line records the presence of the actual capability and also analyses the continuation – this is in line with what happened for ambients above. The remaining lines model the semantics. To understand the formulation it may be helpful to recall the semantics of the in-capability as follows (writing $n : \mu$ to indicate that $\Gamma(n) = \mu$ and writing $\cdot : \mu$ when the ambient name is of no importance for the analysis):



[1] For some applications of Flow Logic to programming languages with higher-order features this need not be the case and then the inference system presentation amounts to an inductive definition rather than the desired co-inductive definition [29,39]; however, this subtle but important point will not be an issue in the present paper where the inductive definition turns out to coincide with the co-inductive definition.

The precondition of the universally quantified implication above recognises the structure depicted to the left of the arrow by querying if the relevant entries already are in $\mathcal{I}$; the conclusion then records the only new structural ingredient depicted to the right of the arrow.

Example 5. Let Γ be given by $\Gamma(\mathsf{p}) = \mathbb{P}$ and $\Gamma(\mathsf{A}) = \Gamma(\mathsf{B}) = \mathbb{S}$ and let $\mathcal{I}$ be given by the second estimate in Example 2:

$$\begin{array}{lcl} \mathcal{I}(\star) & = & \{\mathbb{S}, \mathbb{P}\} \\ \mathcal{I}(\mathbb{S}) & = & \{\mathbb{P}, \mathbb{S}, \mathsf{in}\,\mathbb{S}, \mathsf{out}\,\mathbb{S}, \mathsf{open}\,\mathbb{P}\} \\ \mathcal{I}(\mathbb{P}) & = & \{\mathsf{in}\,\mathbb{S}, \mathsf{out}\,\mathbb{S}\ \} \end{array}$$

Checking that

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}\ ]\ ]\ \mid\ \mathsf{B}\,[\mathsf{open\ p}]$$

involves checking

$$\mathcal{I} \models^{\mathbb{P}}_{\Gamma} \mathsf{in\ B}$$

which holds because $\mathsf{in}\,\mathbb{S} \in \mathcal{I}(\mathbb{P})$ and

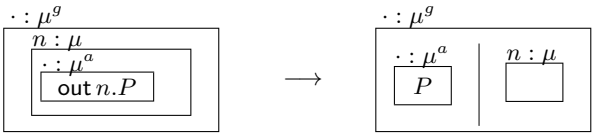
$$\mathsf{in}\,\mathbb{S} \in \mathcal{I}(\mu^a) \wedge \mu^a \in \mathcal{I}(\mu^p) \wedge \mathbb{S} \in \mathcal{I}(\mu^p) \Rightarrow \mu^a \in \mathcal{I}(\mathbb{S})$$

holds for all non-trivial $(\mu^a, \mu^p) \in \{(\mathbb{S}, \star), (\mathbb{S}, \mathbb{S}), (\mathbb{P}, \star), (\mathbb{P}, \mathbb{S})\}$. □

Out-*Capability.* For the out-capability the clause is

$$\begin{array}{lll} \mathcal{I} \models^{\star}_{\Gamma} \mathsf{out}\,n.\ P \text{ iff } & \mathsf{out}\ \mu \in \mathcal{I}(\star)\ \wedge\ \mathcal{I} \models^{\star}_{\Gamma} P\ \wedge & \\ & \forall\ \mu^a, \mu^g : \mathsf{out}\ \mu \in \mathcal{I}(\mu^a) \wedge & \mu^a \text{ has the capability } \mathsf{out}\,\mu \\ & \qquad \mu^a \in \mathcal{I}(\mu)\ \wedge & \mu \text{ is the parent of } \mu^a \\ & \qquad \mu \in \mathcal{I}(\mu^g) & \mu^g \text{ is the grandparent of } \mu^a \\ & \qquad \Rightarrow\ \mu^a \in \mathcal{I}(\mu^g) & \mu^a \text{ may move out of } \mu \\ & \text{where } \mu = \Gamma(n) & \end{array}$$

corresponding to the operational semantics:



Example 6. Continuing Example 5, checking

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}]\ ]\ \mid\ \mathsf{B}\,[\mathsf{open\ p}]$$

involves checking

$$\mathcal{I} \models^{\mathbb{P}}_{\Gamma} \mathsf{out\ A.\ in\ B}$$

which holds because $\mathcal{I} \models^{\mathbb{P}}_{\Gamma} \mathsf{in\ B}$ (see Example 5) and $\mathsf{out}\,\mathbb{S} \in \mathcal{I}(\mathbb{P})$ and

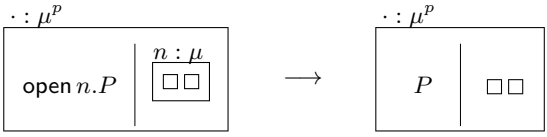
$$\mathsf{out}\,\mathbb{S} \in \mathcal{I}(\mu^a) \wedge \mu^a \in \mathcal{I}(\mathbb{S}) \wedge \mathbb{S} \in \mathcal{I}(\mu^g) \Rightarrow \mu^a \in \mathcal{I}(\mu^g)$$

holds for all $(\mu^a, \mu^g) \in \{(\mathbb{S}, \star), (\mathbb{S}, \mathbb{S}), (\mathbb{P}, \star), (\mathbb{P}, \mathbb{S})\}$. □

Open-*Capability.* For the open-capability the clause is

$$\begin{array}{lll} \mathcal{I} \models^{\star}_{\Gamma} \mathsf{open}\, n.\ P \text{ iff } & \mathsf{open}\ \mu \in \mathcal{I}(\star)\ \wedge\ \mathcal{I} \models^{\star}_{\Gamma} P\ \wedge & \\ & \forall\ \mu^p : \mathsf{open}\ \mu \in \mathcal{I}(\mu^p)\ \wedge & \mu^p \text{ has the capability } \mathsf{open}\,\mu \\ & \quad \mu \in \mathcal{I}(\mu^p) & \mu \text{ is a sibling of } \mathsf{open}\,\mu \\ & \quad \Rightarrow\ \mathcal{I}(\mu) \subseteq \mathcal{I}(\mu^p) & \text{everything in } \mu \text{ may be in } \mu^p \\ & \text{where } \mu = \Gamma(n) & \end{array}$$

corresponding to the operational semantics:



Example 7. Continuing Example 5 and Example 6, checking that

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}\ ]]\ \mid\ \mathsf{B}\,[\mathsf{open\ p}]$$

involves checking

$$\mathcal{I} \models^{\mathbb{S}}_{\Gamma} \mathsf{open\ p}$$

which holds because $\mathsf{open}\,\mathbb{P} \in \mathcal{I}(\mathbb{S})$ and

$$\mathsf{open}\,\mathbb{P} \in \mathcal{I}(\mu^p) \wedge \mathbb{P} \in \mathcal{I}(\mathbb{S}) \Rightarrow \mathcal{I}(\mathbb{P}) \subseteq \mathcal{I}(\mu^p)$$

holds for $\mu^p = \mathbb{S}$. □

This concludes the Flow Logic definition of the judgement $\mathcal{I} \models^{\star}_{\Gamma} P$ in a style close to that of a type system.

Example 8. Ensuring that the analysis estimate $\mathcal{I}$ of Example 5 is an acceptable analysis estimate for *the entire process* $\mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}]\ ]\ \mid\ \mathsf{B}\,[\mathsf{open\ p}]$ amounts to checking that

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}]\ ]\ \mid\ \mathsf{B}\,[\mathsf{open\ p}]$$

This involves checking the clauses for composite processes. The top-level parallel composition gives rise to the two checks that

$$\mathcal{I} \models^{\star}_{\Gamma} \mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A.\ in\ B}]] \quad \text{and} \quad \mathcal{I} \models^{\star}_{\Gamma} \mathsf{B}\,[\mathsf{open\ p}]$$

which, for the first part, leads to the checks that

$$\mathbb{S} \in \mathcal{I}(\star) \quad \text{and} \quad \mathcal{I} \models^{\mathbb{S}}_{\Gamma} \mathsf{p}\,[\mathsf{out\ A.\ in\ B}]$$

In turn, checking the clauses for composite processes leads to the checks of capabilities performed in Example 5, 6, and 7. Performing all the required checks we find that the analysis estimate $\mathcal{I}$ of Example 5 is indeed an acceptable analysis estimate for the 0CFA analysis. □

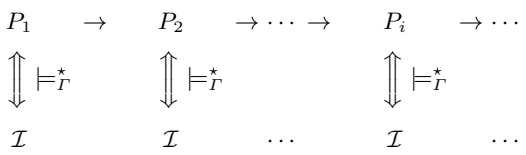


Fig. 3. Subject reduction: the analysis estimate remains acceptable under execution.

Correctness of the 0CFA Analysis. Although the specification of the judgement in the previous subsection was motivated by the transition relation it was not formally linked to it. To do so we take the rather natural approach, familiar from type systems, that the analysis estimate should not only describe the initial configuration in an acceptable way but it must remain acceptable under execution; then we know that all reachable configurations will be described by the analysis estimate.

This is the "subject reduction" approach to correctness; it is illustrated in Figure 3 and formalised by the following theorem:

Theorem 9. *If* $\mathcal{I} \models^\star_\Gamma P$ *and* $P \to^* Q$ *then* $\mathcal{I} \models^\star_\Gamma Q$.

For the proof we first show that the analysis estimate is invariant under the structural congruence:

$$\text{If} \quad P \equiv Q \quad \text{then} \quad \mathcal{I} \models^\star_\Gamma P \quad \text{if and only if} \quad \mathcal{I} \models^\star_\Gamma Q.$$

This amounts to a straightforward induction on the inference of $P \equiv Q$.

Next we prove that the analysis estimate is preserved under the transition relation:

$$\text{If} \quad P \to Q \quad \text{and} \quad \mathcal{I} \models^\star_\Gamma P \quad \text{then} \quad \mathcal{I} \models^\star_\Gamma Q.$$

This amounts to a straightforward induction on the inference of $P \to Q$.

Finally we prove the theorem by a simple numerical induction on the number of steps k in $P \to^k Q$.

Implementation of the 0CFA Analysis. An abstract argument for why there always is a best acceptable analysis estimate borrows from abstract interpretation [19,20,29]. The notion of "best estimate" means "least estimate" since we decided to opt for over-approximation and hence we are looking for a value of $\mathcal{I}$ that contains as few elements as possible. The abstract argument then amounts to the Moore Family result (or model intersection property in model-theoretic terminology):

Theorem 10. *For each* P*, the set* $\{\mathcal{I} \mid \mathcal{I} \models^\star_\Gamma P\}$ *is a Moore family; in other words: for each* P*, if* $\mathcal{Y} \subseteq \{\mathcal{I} \mid \mathcal{I} \models^\star_\Gamma P\}$ *then* $\sqcap\mathcal{Y} \models^\star_\Gamma P$ *where* $(\sqcap\mathcal{Y})(\mu) = \bigcap\{\mathcal{I}(\mu) \mid \mathcal{I} \in \mathcal{Y}\}$.

The proof is by structural induction on P. We are interested in the Moore family property because a Moore family *always* contains a unique least element. Thus it follows that the least analysis estimate can be expressed as $\sqcap\{\mathcal{I} \mid \mathcal{I} \models^\star_\Gamma P\}$ and in the world of type systems this corresponds to each process admitting a single *principal type*.

Table 3. Semantics of ALFP.

$$
\begin{array}{rcl}
(\rho,\sigma) \models R\,(x_1,\ldots,x_k) & \Leftrightarrow & (\sigma\,x_1,\ldots,\sigma\,x_k) \in \rho\,R \\
(\rho,\sigma) \models \neg R\,(x_1,\ldots,x_k) & \Leftrightarrow & (\sigma\,x_1,\ldots,\sigma\,x_k) \notin \rho\,R \\
(\rho,\sigma) \models x = y & \Leftrightarrow & \sigma\,x = \sigma\,y \\
(\rho,\sigma) \models x \neq y & \Leftrightarrow & \sigma\,x \neq \sigma\,y \\
(\rho,\sigma) \models \mathsf{pre}_1 \wedge \mathsf{pre}_2 & \Leftrightarrow & (\rho,\sigma) \models \mathsf{pre}_1 \text{ and } (\rho,\sigma) \models \mathsf{pre}_2 \\
(\rho,\sigma) \models \mathsf{pre}_1 \vee \mathsf{pre}_2 & \Leftrightarrow & (\rho,\sigma) \models \mathsf{pre}_1 \text{ or } (\rho,\sigma) \models \mathsf{pre}_2 \\
(\rho,\sigma) \models \forall x : \mathsf{pre} & \Leftrightarrow & (\rho,\sigma[x \mapsto a]) \models \mathsf{pre} \quad \text{for all } a \in \mathcal{U} \\
(\rho,\sigma) \models \exists x : \mathsf{pre} & \Leftrightarrow & (\rho,\sigma[x \mapsto a]) \models \mathsf{pre} \quad \text{for some } a \in \mathcal{U}
\end{array}
$$

$$
\begin{array}{rcl}
(\rho,\sigma) \models R\,(x_1,\ldots,x_k) & \Leftrightarrow & (\sigma\,x_1,\ldots,\sigma\,x_k) \in \rho\,R \\
(\rho,\sigma) \models \mathbf{1} & \Leftrightarrow & \text{true} \\
(\rho,\sigma) \models \mathsf{clause}_1 \wedge \mathsf{clause}_2 & \Leftrightarrow & (\rho,\sigma) \models \mathsf{clause}_1 \text{ and } (\rho,\sigma) \models \mathsf{clause}_2 \\
(\rho,\sigma) \models \mathsf{pre} \implies \mathsf{clause} & \Leftrightarrow & (\rho,\sigma) \models \mathsf{clause} \text{ whenever } (\rho,\sigma) \models \mathsf{pre} \\
(\rho,\sigma) \models \forall x : \mathsf{clause} & \Leftrightarrow & (\rho,\sigma[x \mapsto a]) \models \mathsf{clause} \quad \text{for all } a \in \mathcal{U}
\end{array}
$$

The ALFP Logic. To actually compute the intended solution in polynomial time we shall follow a rather general and elegant method where the specification is translated into an extension of Horn clauses known as Alternation-free Least Fixed Point Logic (ALFP). This is a first-order logic where the set of formulae (or clauses), clause, and the set of preconditions, pre, are given by the following grammar (subject to a notion of stratification limiting the use of negation):

$$
\begin{array}{lcl}
\mathsf{pre} & ::= & R\,(x_1,\ldots,x_k) \mid \neg R\,(x_1,\ldots,x_k) \mid x = y \mid x \neq y \\
 & & \mathsf{pre}_1 \wedge \mathsf{pre}_2 \mid \mathsf{pre}_1 \vee \mathsf{pre}_2 \mid \forall x : \mathsf{pre} \mid \exists x : \mathsf{pre} \\
\mathsf{clause} & ::= & R\,(x_1,\ldots,x_k) \mid \mathbf{1} \mid \mathsf{clause}_1 \wedge \mathsf{clause}_2 \mid \\
 & & \mathsf{pre} \implies \mathsf{clause} \mid \forall x : \mathsf{clause}
\end{array}
$$

Here R is a k-ary predicate symbol for $k \geq 0$, and $y, x, x_1, \ldots$ denote arbitrary variables, while $\mathbf{1}$ is the always true clause. (Since we shall not use negation in this paper we dispense with explaining the notion of stratification.)

Given a universe finite $\mathcal{U}$ of atomic values and interpretations ρ and σ for predicate symbols and free variables, respectively, the satisfaction relations $(\rho,\sigma) \models \mathsf{pre}$ for pre-conditions and $(\rho,\sigma) \models \mathsf{clause}$ for clauses are defined in a straightforward manner as shown in Table 3. Note that the definitions for pre-conditions and clauses are similar for predicates $R\,(x_1,\ldots,x_k)$), conjunction ($\wedge$), and universal quantification ($\forall$). We view the free variables occurring in a formula as constant symbols or atoms from the finite universe $\mathcal{U}$. Thus, given an interpretation σ of the constant symbols, in the clause clause, we call an interpretation ρ of the predicate symbols a *solution* provided $(\rho,\sigma) \models \mathsf{clause}$.

Implementation Using ALFP. Calculating a least analysis estimate is done by finding the least solution to an ALFP clause that is logically equivalent to the specification of the analysis. The calculation of the least solution ρ is done *automatically* using our Succinct Solver [35]. Recall that being the *least* interpretation means that it contains as few elements as possible while still being acceptable.

In the specification of an analysis we are free to use any mathematical notation, while in the implementation we are limited by what we can express in ALFP. For simple powerset based analyses, such as the ones considered in this paper, the transformation from the specification to ALFP is relatively straightforward. For the analysis considered here the following transformations suffice:

- The mapping $\mathcal{I} : \mathbf{Group} \to \mathcal{P}(\mathbf{Group} \cup \mathbf{Cap})$ is encoded as a *binary predicate* of sort $\mathbf{Group} \times (\mathbf{Group} \cup \mathbf{Cap})$.
- Correspondingly, set membership such as $\mu' \in \mathcal{I}(\mu)$ is written as $\mathcal{I}(\mu, \mu')$.
- Subset relations such as $\mathcal{I}(\mu) \subseteq \mathcal{I}(\mu')$ are written by explicitly quantifying the elements in the first set: $\forall u : \mathcal{I}(\mu, u) \Rightarrow \mathcal{I}(\mu', u)$.
- All groups μ and group capabilities $\mathsf{in}\,\mu, \mathsf{out}\,\mu$, and $\mathsf{open}\,\mu$ are elements in the universe $\mathcal{U}$; for a given process this universe will be finite.

It is straightforward to establish a formal relationship between the specification and the implementation by giving a mapping (actually an isomorphism) between the two representations of $\mathcal{I}$ and showing that the specification and the implementation are logically equivalent under this mapping.

We apply the above encodings systematically to the specification of the analysis thus getting a new formulation from which we can generate ALFP clauses for any given process P. The analysis of composite processes remains unchanged except for the analysis of the ambient construct, which is changed into:

$$\mathcal{I} \models^{\star}_{\Gamma} n\,[P] \;\text{ iff }\; \mathcal{I}(\star, \mu) \;\wedge\; \mathcal{I} \models^{\mu}_{\Gamma} P \quad \text{where } \mu = \Gamma(n)$$

That is, the set membership is now written $\mathcal{I}(\star, \mu)$. Similarly, the analysis of the capabilities are changed and in the case of $\mathsf{open}\,\mu$ (where subset is used) the clause becomes:

$$\begin{aligned}\mathcal{I} \models^{\star}_{\Gamma} \mathsf{open}\,n.\ P \text{ iff } \; & \mathcal{I}(\star, \mathsf{open}\ \mu) \wedge \mathcal{I} \models^{\star}_{\Gamma} P \ \wedge \\ & \forall\ \mu^p : \mathcal{I}(\mu^p, \mathsf{open}\ \mu) \wedge \\ & \qquad \mathcal{I}(\mu^p, \mu) \\ & \qquad \Rightarrow \forall u : \mathcal{I}(\mu, u) \Rightarrow \mathcal{I}(\mu^p, u) \\ & \text{where } \mu = \Gamma(n)\end{aligned}$$

The rules for in- and out-capabilities are obtained in an analogous way.

Example 11. Finding the least analysis estimate for the 0CFA analysis of the process

$$\mathsf{A}\,[\mathsf{p}\,[\mathsf{out\ A}.\ \mathsf{in\ B}]] \mid \mathsf{B}\,[\mathsf{open\ p}]$$

with Γ given by $\Gamma(\mathsf{p}) = \mathbb{P}$ and $\Gamma(\mathsf{A}) = \Gamma(\mathsf{B}) = \mathbb{S}$ now amounts to finding the solution to the ALFP clause:

$$\begin{aligned}
&\mathcal{I}(\star, \mathbb{S}) \wedge && \text{Ambient A}\\
&\mathcal{I}(\mathbb{S}, \mathbb{P}) \wedge && \text{Ambient p}\\
&\mathcal{I}(\mathbb{P}, \mathsf{out}\,\mathbb{S}) \wedge && \text{Capability out A}\\
&(\forall \mu^a, \mu^g : \mathcal{I}(\mu^a, \mathsf{out}\,\mathbb{S}) \wedge \mathcal{I}(\mathbb{S}, \mu^a) \wedge \mathcal{I}(\mu^g, \mathbb{S}) \\
&\qquad \Rightarrow \mathcal{I}(\mu^g, \mu^a)) \wedge
\end{aligned}$$

$$
\begin{array}{ll}
\mathcal{I}(\mathbb{P}, \mathsf{in}\,\mathbb{S}) \wedge & \text{Capability } \mathsf{in}\,\mathsf{B} \\
(\forall \mu^a, \mu^p : \mathcal{I}(\mu^a, \mathsf{in}\,\mathbb{S}) \wedge \mathcal{I}(\mu^p, \mu^a) \wedge \mathcal{I}(\mu^p, \mathbb{S}) & \\
\quad \Rightarrow \mathcal{I}(\mathbb{S}, \mu^a)) \wedge & \\
\mathcal{I}(\star, \mathbb{S}) \wedge & \text{Ambient } \mathsf{B} \\
\mathcal{I}(\mathbb{S}, \mathsf{open}\,\mathbb{P}) \wedge & \text{Capability } \mathsf{open\,p} \\
(\forall \mu^p : \mathcal{I}(\mu^p, \mathsf{open}\,\mathbb{P}) \wedge \mathcal{I}(\mu^p, \mathbb{P}) & \\
\quad \Rightarrow \forall u : \mathcal{I}(\mathbb{P}, u) \Rightarrow \mathcal{I}(\mu^p, u)) &
\end{array}
$$

The resulting least solution $\mathcal{I}$ which satisfies the clause is:

$$
\begin{array}{ll}
\mathcal{I} : (\star, \mathbb{S}), (\mathbb{S}, \mathbb{P}), & \text{from ambients} \\
\quad (\mathbb{P}, \mathsf{out}\,\mathbb{S}), (\star, \mathbb{P}), & \text{from } \mathsf{out} \\
\quad (\mathbb{P}, \mathsf{in}\,\mathbb{S}), (\mathbb{S}, \mathbb{S}), & \text{from } \mathsf{in} \\
\quad (\mathbb{S}, \mathsf{open}\,\mathbb{P}), (\mathbb{S}, \mathsf{out}\,\mathbb{S}), (\mathbb{S}, \mathsf{in}\,\mathbb{S}) & \text{from } \mathsf{open}
\end{array}
$$

This corresponds to the solution displayed in Example 5. □

The solution of an ALFP clause can be found in polynomial time in the size of the universe i.e. in the number of groups and capabilities. This complexity is due to a generalisation [35] of a meta-complexity result for Horn Clauses by McAllester [25], which states that:

- The time needed to *compute* a solution is asymptotically the same as the time needed to *check* the validity of an estimate.
- The degree of the complexity polynomial is dominated by one plus the nesting depth of quantifiers occurring in the clause.

Consequently, the complexity bound can sometimes be improved by reformulating the clause to reduce the amount of quantifier nesting. Rather than improving formulae using a general transformation scheme, like the use of *tiling* to reduce quantifier nesting [36], or *automatically estimating* the run-time [32], we take the more pragmatic, and more precise, approach of estimating the run-time empirically [7], and use this as a basis for transforming the clause so as to improve its running time [7].

A result of such an experiment is shown in Figure 4. Here the analysis has been run on a number of processes with the same overall structure where a packet is routed through a grid of $m \times m$ sites. The actual time that the Succinct Solver spends on computing a solution is plotted against the size of the process.

The plot is shown using logarithmic scales on both axes so that a power function shows up as a straight line. A crude least-squares estimate of the degree of the complexity polynomial is displayed in the legend of the plot and we see that the solving times are *linear* in the size of the process being analysed. This is typical for most processes, though the analysis in some cases runs in time that is quadratic in the size of the process.

Remark 12. We already said that for ALFP the time needed to *compute* the least solution is asymptotically the same as the time needed to *check* the acceptability of an estimate. Elaborating on the analogy to type systems explained above, one could then coin the slogan:

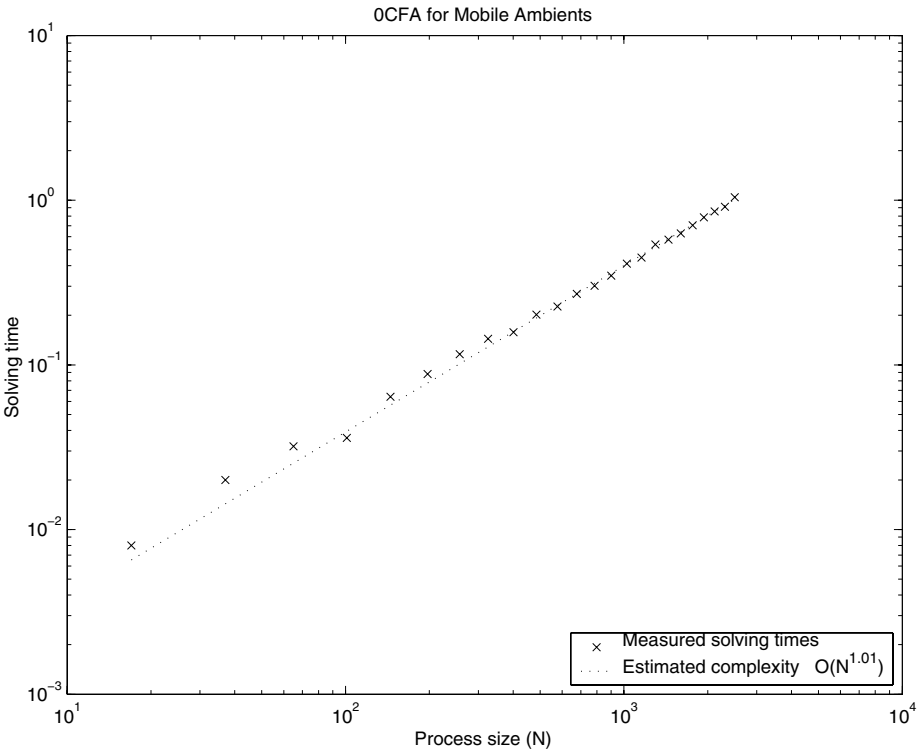


Fig. 4. Estimating the complexity empirically.

> ALFP-based Flow Logic studies a class of simple Type Systems where *type checking* and *type inference* have the same asymptotic complexity.

In the absence of principal types this is quite different from type systems based on subtyping where *type checking* usually takes polynomial time but where *type inference* often would seem to require non-deterministic polynomial time (in practise exponential time) due to the need to search for the right types. □

2.3 Crossing Control and Opening Control

The analysis not only approximates the hierarchical structure of the ambients but also the access operations that an ambient may possess. This facilitates validating the following security properties [12]:

- *Crossing control:* may an ambient m cross the boundary of another ambient n either by entering it (using in-capabilities) or by exiting it (using out-capabilities)?
- *Opening control:* may an ambient n be dissolved by another ambient m (using open-capabilities)?

In each case we proceed as follows:

- First, we describe the desired property *dynamically*, i.e. we express it using the concepts and notation of the reduction semantics.
- Second, we describe the property *statically*, i.e. we re-express it using the concepts and notation of the static analysis.
- Third, we show the semantic correctness of these formulations: that the static formulation of the property implies the dynamic formulation.
- Finally, we argue that the test can be performed by means of the techniques used for implementing the analysis, which in our case means that the static properties can be determined in polynomial time.

Crossing Control. The dynamic notion amounts to saying that an ambient n can cross the ambient n' during the execution of a process P whenever in some reachable configuration n executes the $\mathsf{in}\, n'$ or the $\mathsf{out}\, n'$ capability. This can be reformulated in terms of groups:

Definition 13 (Dynamic Notion of Crossing). *Ambients of group μ* **can cross** *ambients of group μ' during the execution of P* whenever

1. $P \rightarrow^* Q$,
2. *some ambient n in Q contains an executable capability* $\mathsf{in}\, n'$ *or an executable capability* $\mathsf{out}\, n'$, *and*
3. *n is of group μ and n' is of group μ'.*

We could choose to define the dynamic notion both with and without groups but we shall focus on the former since it more directly relates to the static notion studied below.

The static notion is expressed in terms of the 0CFA analysis. It amounts to saying that ambients of group μ may cross ambients of group μ' during the execution of P whenever the precondition in the clause for in-capabilities is satisfied or the precondition in the clause for out-capabilities is satisfied. To express this in a succinct manner we decide to introduce an"observation predicate", named $\mathcal{D}$ for dynamics, to keep track of the capabilities recorded by $\mathcal{I}$ that may actually execute according to the analysis. We let $\mathcal{D}$ be a mapping $\mathcal{D} : \mathbf{Group} \rightarrow \mathcal{P}(\mathbf{Cap})$ and modify the clauses for in and out to read:

$$
\begin{array}{rl}
(\mathcal{I},\mathcal{D}) \models^\star_\Gamma \mathsf{in}\, n.\ P \text{ iff } & \mathsf{in}\ \mu \in \mathcal{I}(\star)\ \wedge\ (\mathcal{I},\mathcal{D}) \models^\star_\Gamma P\ \wedge \\
& \forall\ \mu^a, \mu^p : \mathsf{in}\ \mu \in \mathcal{I}(\mu^a)\ \wedge \\
& \qquad \mu^a \in \mathcal{I}(\mu^p)\ \wedge \\
& \qquad \mu \in \mathcal{I}(\mu^p) \\
& \quad \Rightarrow\ \mu^a \in \mathcal{I}(\mu)\ \wedge\ \mathsf{in}\ \mu \in \mathcal{D}(\mu^a) \\
& \text{where } \mu = \Gamma(n)
\end{array}
$$

$$
\begin{array}{rl}
(\mathcal{I},\mathcal{D}) \models^\star_\Gamma \mathsf{out}\, n.\ P \text{ iff } & \mathsf{out}\ \mu \in \mathcal{I}(\star)\ \wedge\ (\mathcal{I},\mathcal{D}) \models^\star_\Gamma P\ \wedge \\
& \forall\ \mu^a, \mu^g : \mathsf{out}\ \mu \in \mathcal{I}(\mu^a) \wedge \\
& \qquad \mu^a \in \mathcal{I}(\mu)\ \wedge \\
& \qquad \mu \in \mathcal{I}(\mu^g) \\
& \quad \Rightarrow\ \mu^a \in \mathcal{I}(\mu^g)\ \wedge\ \mathsf{out}\ \mu \in \mathcal{D}(\mu^a) \\
& \text{where } \mu = \Gamma(n)
\end{array}
$$

Using the information in the "observation predicate" $\mathcal{D}$ the static notion of what it means for an ambient to cross the boundary of another ambient can be defined as follows:

Definition 14 (Static Notion of Crossing). *Ambients of group* μ **possibly may cross** *ambients of group* μ' *during the execution of* P whenever

$$\mathsf{in}\,\mu' \in \mathcal{D}(\mu) \quad \vee \quad \mathsf{out}\,\mu' \in \mathcal{D}(\mu)$$

for the least $\mathcal{I}$ *and* $\mathcal{D}$ *such that* $(\mathcal{I}, \mathcal{D}) \models^{\star}_{\Gamma} P$.

This static condition is checkable in polynomial time.

Example 15. With respect to Γ and $\mathcal{I}$ as displayed in Example 5 the least estimate for the modified analysis will produce a relation $\mathcal{D}$ that contains exactly the same capabilities as recorded in $\mathcal{I}$; i.e. $\forall \mu : \mathcal{D}(\mu) = \mathcal{I}(\mu) \cap \mathbf{Cap}$. Hence the analysis can be used to validate (where "will never" is the negation of "possibly may"):

- Ambients of group $\mathbb{P}$ possibly may cross ambients in group $\mathbb{S}$;
  because $\mathsf{in}\,\mathbb{S} \in \mathcal{D}(\mathbb{P}) \ \vee \ \mathsf{out}\,\mathbb{S} \in \mathcal{D}(\mathbb{P})$.
- Ambients in group $\mathbb{S}$ will never cross ambients in group $\mathbb{P}$;
  because $\mathsf{in}\,\mathbb{P} \notin \mathcal{D}(\mathbb{S}) \wedge \ \mathsf{out}\,\mathbb{P} \notin \mathcal{D}(\mathbb{S})$.

It is interesting to observe that a more precise analysis is needed to validate that ambients of group $\mathbb{S}$ will never cross ambients in group $\mathbb{S}$ since we do *not* have that $\mathsf{in}\,\mathbb{S} \notin \mathcal{D}(\mathbb{S}) \wedge \mathsf{out}\,\mathbb{S} \notin \mathcal{D}(\mathbb{S})$. And indeed, we also have $\mathbb{S} \in \mathcal{I}(\mathbb{S})$ indicating that as far as the analysis can see, some ambient of group $\mathbb{S}$ may turn up inside some ambient of group $\mathbb{S}$. □

The correctness of the static test with respect to the dynamic semantics is formally expressed as follows:

Theorem 16 (Crossing Control).

1. If *ambients of group* μ **can cross** *ambients of group* μ' *during the execution of* P then *ambients of group* μ **possibly may cross** *ambients of group* μ' *during the execution of* P.
2. If *ambients of group* μ **will never cross** *ambients of group* μ' *during the execution of* P then *ambients of group* μ **cannot cross** *ambients of group* μ' *during the execution of* P.

The proposition is a corollary of the subject reduction result (Theorem 9); also note that the second statement is the contrapositive version of the first statement (and hence that they are logically equivalent).

Opening Control. The dynamic notion amounts to saying that an ambient n can open the ambient n' during the execution of P whenever some n executes the $\mathsf{open}\,n'$ capability in some reachable configuration. Again we define the notion in terms of groups.

Definition 17 (Dynamic Notion of Opening). *Ambients of group μ* **can open** *ambients of group μ' during the execution of P* whenever

1. $P \rightarrow^* Q$,
2. *some ambient n in Q contains an executable capability* $\mathsf{open}\, n'$, *and*
3. *n is of group μ and n' is of group μ'.*

The static notion is once again expressed in terms of the 0CFA analysis. It amounts to saying that ambients of group μ may open ambients in group μ' whenever the precondition in the clause for open-capabilities is satisfied. As before we use a modified clause for extracting the "executable" capabilities in $\mathcal{D}$:

$$\begin{array}{l}(\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} \mathsf{open}\, n.\ P \text{ iff } \mathsf{open}\ \mu \in \mathcal{I}(\star)\ \wedge\ (\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} P\ \wedge \\ \qquad\qquad \forall\ \mu^p : \mathsf{open}\ \mu \in \mathcal{I}(\mu^p)\ \wedge \\ \qquad\qquad\qquad \mu \in \mathcal{I}(\mu^p) \\ \qquad\qquad\quad \Rightarrow \mathcal{I}(\mu) \subseteq \mathcal{I}(\mu^p) \wedge \mathsf{open}\, \mu \in \mathcal{D}(\mu^p) \\ \qquad\qquad \text{where } \mu = \Gamma(n)\end{array}$$

and the static property can be defined accordingly:

Definition 18 (Static Notion of Opening). *Ambients of group μ* **possibly may open** *ambients of group μ' during the execution of P* whenever

$$\mathsf{open}\, \mu' \in \mathcal{D}(\mu)$$

for the least $\mathcal{I}$ and $\mathcal{D}$ such that $(\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} P$.

As before the condition is checkable in polynomial time.

Example 19. With respect to Γ, $\mathcal{I}$ and $\mathcal{D}$ as given in Examples 5 and 15, respectively, the analysis can be used to validate that ambients of group $\mathbb{S}$ possibly may open ambients in group $\mathbb{P}$, because $\mathsf{open}\, \mathbb{P} \in \mathcal{D}(\mathbb{S})$, and that ambients in group $\mathbb{P}$ will never open any ambients, because $\forall \mu : \mathsf{open}\, \mu \notin \mathcal{D}(\mathbb{P})$. □

Theorem 20 (Opening Control).

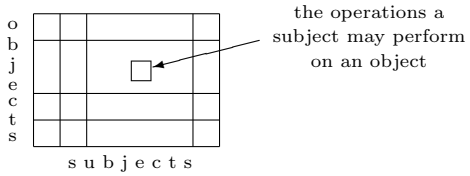
1. If *ambients of group μ* **can open** *ambients in group μ' during the execution of P* then *ambients of group μ* **possibly may open** *ambients in group μ' during the execution of P.*
2. If *ambients of group μ* **will never open** *ambients in group μ' during the execution of P* then *ambients of group μ* **cannot open** *ambients in group μ' during the execution of P.*

As before this is a corollary of the subject reduction result (Theorem 9).

3 Discretionary Access Control

The notion of discretionary access control originates from operating systems where it is used to define a reference monitor for governing the access operations (typically read, write and execute) that active subjects (typically programs or

users) can perform on passive objects (typically files or external devices). The reference monitor is then implemented as part of the operating system. Although traditionally implemented as access control lists, often based on grouping users into three layers (the user, a group of users, all users), conceptually the specification of access control takes the form of a matrix [22]:



When adapting discretionary access control to Mobile Ambients we should rethink the concepts of subject, object and access operation. It seems very natural to let the access operations be the basic capabilities (in, out and open) of Mobile Ambients since the notions of read, write and execute are indeed the basic operations of a traditional operating system. Then subjects and objects will both be ambients; the subject will be the ambient containing the capability and the object the other ambient involved (typically the one being moved into or being moved out of).

Safe Ambients [24] extend Mobile Ambients to deal with discretionary access control. Since it is not in the distributed nature of Mobile Ambients to have a single global access control matrix it is implemented as access rights, or co-capabilities, placed inside the objects. Syntactically this leads to modifying the syntax of Mobile Ambients as follows:

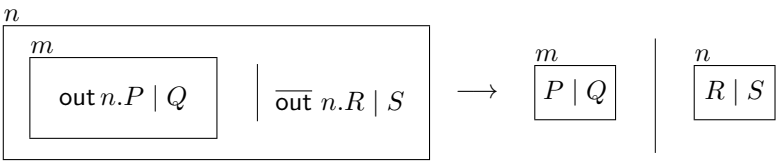
$$P ::= \cdots \text{ as before } \cdots$$

$$\begin{array}{rll} M ::= & \mathsf{in}\ n \mid \mathsf{out}\ n \mid \mathsf{open}\ n & \text{capabilities} \approx \text{access operations} \\ \mid & \overline{\mathsf{in}}\ n \mid \overline{\mathsf{out}}\ n \mid \overline{\mathsf{open}}\ n & \text{co-capabilities} \approx \text{access rights} \end{array}$$

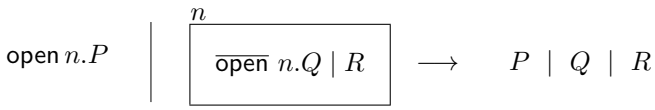
A transition only takes place if a capability of the subject is matched by a corresponding co-capability in the object:

- If m wants to move into n then n should be willing to let ambients enter; i.e. n must have the co-capability $\overline{\mathsf{in}}\ n$:

- If m wants to move out of n then n should be willing to let ambients leave; i.e. n must have the co-capability $\overline{\mathsf{out}}\ n$:



– If m wants to dissolve n then n should be willing to be dissolved; i.e. n must have the co-capability $\overline{\mathsf{open}}\ n$:



This amounts to integrating the reference monitor into the semantics i.e. the transition relation.

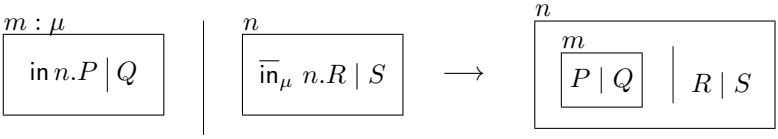
3.1 Syntax and Semantics of Discretionary Ambients

Discretionary Ambients goes one step further in giving an account of discretionary access control that is as refined as illustrated by the access control matrix above. We do so by augmenting co-capabilities with a subscript indicating the group of ambients permitted to perform the corresponding capability:

$$P ::= (\nu\mu)P \mid (\nu n : \mu)P \mid \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid n[P] \mid M.P$$
$$M ::= \mathsf{in}\ n \mid \mathsf{out}\ n \mid \mathsf{open}\ n \mid \overline{\mathsf{in}}_\mu n \mid \overline{\mathsf{out}}_\mu n \mid \overline{\mathsf{open}}_\mu n$$

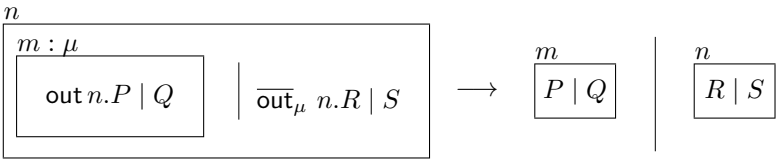
Hence the basic transitions need to be determined relative to a type environment Γ mapping ambient names to groups; below we write $n : \mu$ to indicate that $\Gamma(n) = \mu$.

– For the in-capability we have



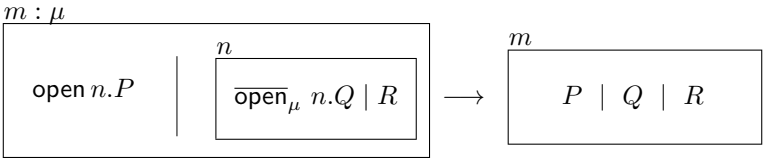
so n is willing to let ambients of group μ enter.

– For the out-capability we have



so n is willing to let ambients of group μ leave.

– For the open-capability we have



so n is willing to be dissolved within ambients of group μ.

Table 4. Transition relation for Discretionary Ambients.

$$\frac{\Gamma[\mu \mapsto \diamond] \vdash P \to Q}{\Gamma \vdash (\nu\mu)P \to (\nu\mu)Q} \qquad \frac{\Gamma[n \mapsto \mu] \vdash P \to Q}{\Gamma \vdash (\nu n : \mu)P \to (\nu n : \mu)Q} \quad \text{if } \mu \in \mathrm{dom}(\Gamma)$$

$$\frac{\Gamma \vdash P \to Q}{\Gamma \vdash n[P] \to n[Q]} \qquad \Gamma \vdash m[\mathsf{in}\ n.\, P \mid Q] \mid n[\overline{\mathsf{in}}_\mu n.\, R \mid S] \to n[m[P \mid Q] \mid R \mid S] \quad \text{if } \Gamma(m) = \mu$$

$$\frac{\Gamma \vdash P \to Q}{\Gamma \vdash P \mid R \to Q \mid R} \qquad \Gamma \vdash n[m[\mathsf{out}\ n.\, P \mid Q] \mid \overline{\mathsf{out}}_\mu n.\, R \mid S] \to m[P \mid Q] \mid n[R \mid S] \quad \text{if } \Gamma(m) = \mu$$

$$\frac{P \equiv P' \quad \Gamma \vdash P' \to Q' \quad Q' \equiv Q}{\Gamma \vdash P \to Q} \qquad \Gamma \vdash m[\mathsf{open}\ n.\, P \mid n[\overline{\mathsf{open}}_\mu n.\, Q \mid R]] \to m[P \mid Q \mid R] \quad \text{if } \Gamma(m) = \mu$$

The semantics of Discretionary Ambients is a straightforward extension of the semantics of Mobile Ambients. It consists of the structural congruence relation, $P \equiv Q$, defined in Table 1 and the transition relation, $\Gamma \vdash P \to Q$, defined in Table 4. Here Γ is a type environment mapping names to groups and groups to the special token $\diamond$.

Example 21. We may express the process of Figure 1 in Discretionary Ambients as follows:

$$\mathsf{A}[\mathsf{p}[\mathsf{out}\ \mathsf{A}.\mathsf{in}\ \mathsf{B}.\overline{\mathsf{open}}_{\mathbb{S}}\ \mathsf{p}] \mid \overline{\mathsf{out}}_{\mathbb{P}}\ \mathsf{A}] \mid \mathsf{B}[\overline{\mathsf{in}}_{\mathbb{P}}\ \mathsf{B}.\mathsf{open}\ \mathsf{p}]$$

where we assume that $\Gamma(\mathsf{A}) = \Gamma(\mathsf{B}) = \mathbb{S}$ and $\Gamma(\mathsf{p}) = \mathbb{P}$ i.e. that the sites A and B are in the group $\mathbb{S}$ and the packet p is in the group $\mathbb{P}$. The packet may move out of A since it has the capability $\mathsf{out}\ \mathsf{A}$ and furthermore A grants it the right to do so because it has the co-capability $\overline{\mathsf{out}}_{\mathbb{P}}\ \mathsf{A}$ (and exploiting that p is in the group $\mathbb{P}$). So in the first step the system may evolve into:

$$\mathsf{A}[\,] \mid \mathsf{p}[\mathsf{in}\ \mathsf{B}.\overline{\mathsf{open}}_{\mathbb{S}}\ \mathsf{p}] \mid \mathsf{B}[\overline{\mathsf{in}}_{\mathbb{P}}\ \mathsf{B}.\mathsf{open}\ \mathsf{p}]$$

Now p has the capability to enter B and B has the co-capability to let it do so; the system becomes:

$$\mathsf{A}[\,] \mid \mathsf{B}[\mathsf{p}[\overline{\mathsf{open}}_{\mathbb{S}}\ \mathsf{p}] \mid \mathsf{open}\ \mathsf{p}]$$

In the final step B has the capability to open p and p grants it the right to do so since B is in the group $\mathbb{S}$ and we then obtain:

$$\mathsf{A}[\,] \mid \mathsf{B}[\,]$$

as the final configuration. □

Remark 22. The classical Mobile Ambients do not support access control: an object has no way of restricting which operations the subjects may perform on it. Safe Ambients [24] allows to model a very rudimentary form of access control

as the objects may use the co-capabilities $\overline{\mathsf{in}}\, n$, $\overline{\mathsf{out}}\, n$ and $\overline{\mathsf{open}}\, n$ to control which operations they engage in. However, the possession of one of these co-capabilities gives access to *any* subject with a corresponding capability; there is no way of allowing some subjects but not others to perform the operation. Discretionary Ambients models the more general form of access control corresponding to the classical developments because the co-capabilities put restrictions on the subjects allowed to perform the operations.

The difference can be illustrated for the running example where the access control matrices could be viewed as being:

Safe Ambients

object \ subject	A	B	p
A	$\overline{\mathsf{out}}$ A	$\overline{\mathsf{out}}$ A	$\overline{\mathsf{out}}$ A
B	$\overline{\mathsf{in}}$ B	$\overline{\mathsf{in}}$ B	$\overline{\mathsf{in}}$ B
p	$\overline{\mathsf{open}}$ p	$\overline{\mathsf{open}}$ p	$\overline{\mathsf{open}}$ p

Discretionary Ambients

object \ subject	A : $\mathbb{S}$	B : $\mathbb{S}$	p : $\mathbb{P}$
A	–	–	$\overline{\mathsf{out}}_{\mathbb{P}}$ A
B	–	–	$\overline{\mathsf{in}}_{\mathbb{P}}$ B
p	$\overline{\mathsf{open}}_{\mathbb{S}}$ p	$\overline{\mathsf{open}}_{\mathbb{S}}$ p	–

Note that for Safe Ambients columns must necessarily be equal.

Compared to the classical setting the access control matrices in Discretionary Ambients are much more dynamic structures that may evolve as the process executes. This is due to the fact that co-capabilities vanish once they have been used. If we want to model the classical setting more faithfully we should therefore always use co-capabilities that are individual threads and prefixed with the replication operator: e.g. $!\,\overline{\mathsf{in}}_{\mathbb{P}}\ \mathsf{B}$ will continue to grant subjects of group $\mathbb{P}$ permission to enter the ambient B as many times as needed. □

3.2 Adapting the 0CFA Analysis to Discretionary Ambients

To adapt the 0CFA analysis to deal with Discretionary (or Safe) Ambients we must modify the functionality of $\mathcal{I}$ to record

- as before: which ambient groups may be inside an ambient in group μ,
- as before: which access operations (capabilities) may be possessed by an ambient in group μ (as subject), and
- additionally: which access rights (co-capabilities) may be possessed by an ambient in group μ (as object).

Hence we shall use

$$\mathcal{I} : \mathbf{Group} \to \mathcal{P}(\mathbf{Group} \cup \mathbf{Cap} \cup \overline{\mathbf{Cap}})$$

where capabilities and co-capabilities are given by:

$$\begin{array}{lll} \text{capabilities} & m \in \mathbf{Cap} & m ::= \mathsf{in}\ \mu \mid \mathsf{out}\ \mu \mid \mathsf{open}\ \mu \\ \text{co-capabilities} & \overline{m} \in \overline{\mathbf{Cap}} & \overline{m} ::= \overline{\mathsf{in}}_{\mu'}\ \mu \mid \overline{\mathsf{out}}_{\mu'}\ \mu \mid \overline{\mathsf{open}}_{\mu'}\ \mu \end{array}$$

We shall find it useful also to incorporate the observations $\mathcal{D}$ as discussed in Subsection 2.3 and thus have:

$$\mathcal{D} : \mathbf{Group} \to \mathcal{P}(\mathbf{Cap} \cup \overline{\mathbf{Cap}})$$

Specification of the Adapted 0CFA Analysis. It is straightforward to adapt the specification of the 0CFA analysis from Section 2.2 to deal with Discretionary Ambients. In particular no changes are needed for the analysis of the composite processes.

For co-capabilities we simply record the presence of the co-capability much as was the case for ambients:

$$(\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} \overline{\mathsf{in}}_{\mu^a}\ n.\ P \ \text{ iff } \ \overline{\mathsf{in}}_{\mu^a}\mu \in \mathcal{I}(\star) \wedge (\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} P \quad \text{where } \mu = \Gamma(n)$$

$$(\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} \overline{\mathsf{out}}_{\mu^a}\ n.\ P \ \text{ iff } \ \overline{\mathsf{out}}_{\mu^a}\mu \in \mathcal{I}(\star) \wedge (\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} P \quad \text{where } \mu = \Gamma(n)$$

$$(\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} \overline{\mathsf{open}}_{\mu^p}\ n.\ P \ \text{ iff } \ \overline{\mathsf{open}}_{\mu^p}\mu \in \mathcal{I}(\star) \wedge (\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} P \quad \text{where } \mu = \Gamma(n)$$

For capabilities we need to add one conjunct in each precondition that ensures that the capability is matched by a corresponding co-capability:

$$\begin{aligned}(\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} \mathsf{in}\, n.\ P \text{ iff } & \mathsf{in}\ \mu \in \mathcal{I}(\star)\ \wedge\ (\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} P\ \wedge \\ & \forall\ \mu^a,\mu^p : \mathsf{in}\ \mu \in \mathcal{I}(\mu^a)\ \wedge\ \mu^a \in \mathcal{I}(\mu^p)\ \wedge\ \mu \in \mathcal{I}(\mu^p)\ \wedge \\ & \qquad \overline{\mathsf{in}}_{\mu^a}\mu \in \mathcal{I}(\mu) \quad \mu \text{ provides the access right to } \mu^a \\ & \quad \Rightarrow\ \mu^a \in \mathcal{I}(\mu)\ \wedge\ \mathsf{in}\ \mu \in \mathcal{D}(\mu^a) \wedge \overline{\mathsf{in}}_{\mu^a}\mu \in \mathcal{D}(\mu) \\ & \text{where } \mu = \Gamma(n)\end{aligned}$$

$$\begin{aligned}(\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} \mathsf{out}\, n.\ P \text{ iff } & \mathsf{out}\ \mu \in \mathcal{I}(\star)\ \wedge\ (\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} P\ \wedge \\ & \forall\ \mu^a,\mu^g : \mathsf{out}\ \mu \in \mathcal{I}(\mu^a) \wedge \mu^a \in \mathcal{I}(\mu) \wedge \mu \in \mathcal{I}(\mu^g)\ \wedge \\ & \qquad \overline{\mathsf{out}}_{\mu^a}\mu \in \mathcal{I}(\mu) \quad \mu \text{ provides the access right to } \mu^a \\ & \quad \Rightarrow\ \mu^a \in \mathcal{I}(\mu^g)\ \wedge \\ & \qquad \mathsf{out}\ \mu \in \mathcal{D}(\mu^a) \wedge \overline{\mathsf{out}}_{\mu^a}\mu \in \mathcal{D}(\mu) \\ & \text{where } \mu = \Gamma(n)\end{aligned}$$

$$\begin{aligned}(\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} \mathsf{open}\, n.\ P \text{ iff } & \mathsf{open}\ \mu \in \mathcal{I}(\star)\ \wedge\ (\mathcal{I},\mathcal{D}) \models^{\star}_{\Gamma} P\ \wedge \\ & \forall\ \mu^p : \mathsf{open}\ \mu \in \mathcal{I}(\mu^p)\ \wedge\ \mu \in \mathcal{I}(\mu^p)\ \wedge \\ & \qquad \overline{\mathsf{open}}_{\mu^p}\mu \in \mathcal{I}(\mu) \quad \mu \text{ provides the access right to } \mu^p \\ & \quad \Rightarrow\ \mathcal{I}(\mu) \subseteq \mathcal{I}(\mu^p) \wedge \\ & \qquad \mathsf{open}\ \mu \in \mathcal{D}(\mu^p) \wedge \overline{\mathsf{open}}_{\mu^p}\mu \in \mathcal{D}(\mu) \\ & \text{where } \mu = \Gamma(n)\end{aligned}$$

Here $\mathcal{D}$ records capabilities and co-capabilities whenever they may be executed.

Example 23. Continuing Example 21 we take Γ to be as before (i.e. $\Gamma(\mathsf{p}) = \mathbb{P}$ and $\Gamma(\mathsf{A}) = \Gamma(\mathsf{B}) = \mathbb{S}$) and obtain the following best estimates of $\mathcal{I}$ and $\mathcal{D}$ of the 0CFA given above:

$$\begin{array}{lcl}
\mathcal{I}(\star) & = & \{\mathbb{S}, \mathbb{P}\} \\
\mathcal{I}(\mathbb{S}) & = & \{\mathbb{P}, \mathsf{in}\,\mathbb{S}, \mathsf{out}\,\mathbb{S}, \mathsf{open}\,\mathbb{P}, \overline{\mathsf{in}}_{\mathbb{P}}\ \mathbb{S}, \overline{\mathsf{out}}_{\mathbb{P}}\ \mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\ \mathbb{P}\} \\
\mathcal{I}(\mathbb{P}) & = & \{\mathsf{in}\,\mathbb{S}, \mathsf{out}\,\mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\ \mathbb{P}\}
\end{array}$$

$$\begin{array}{lcl}
\mathcal{D}(\star) & = & \emptyset \\
\mathcal{D}(\mathbb{S}) & = & \{\mathsf{open}\,\mathbb{P}, \overline{\mathsf{in}}_{\mathbb{P}}\ \mathbb{S}, \overline{\mathsf{out}}_{\mathbb{P}}\ \mathbb{S}\} \\
\mathcal{D}(\mathbb{P}) & = & \{\mathsf{in}\,\mathbb{S}, \mathsf{out}\,\mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\ \mathbb{P}\}
\end{array}$$

The notion of crossing control from Section 2.3 can easily be adapted to the setting of Discretionary Ambients. Now the analysis finds that ambients of group $\mathbb{S}$ will never cross ambients of group $\mathbb{S}$ since $\mathsf{in}\,\mathbb{S} \notin \mathcal{D}(\mathbb{S})$ and $\mathsf{out}\,\mathbb{S} \notin \mathcal{D}(\mathbb{S})$. This is unlike what was the case for the analysis of Mobile Ambients in Example 15. □

Remark 24. Clearly the analysis can be simplified to deal with Safe Ambients, rather than Discretionary Ambients, although with a loss in precision. This is done by "ignoring" the groups in the co-capabilities in the specification of the analysis above. Continuing Example 23 we can express the system in Safe Ambients by omitting the subscripts to the co-capabilities:

$$\mathsf{A}\,[\,\mathsf{p}\,[\,\mathsf{out}\,\mathsf{A}.\ \mathsf{in}\,\mathsf{B}.\ \overline{\mathsf{open}}\ \mathsf{p}\,]\ \mid\ \overline{\mathsf{out}}\ \mathsf{A}\,]\ \mid\ \mathsf{B}\,[\,\overline{\mathsf{in}}\ \mathsf{B}.\ \mathsf{open}\,\mathsf{p}\,]$$

When we apply the (modified) analysis from above to this process, however, the best analysis estimate will resemble the one found for Mobile Ambients in Example 15 and is no longer able to ensure that ambients of group $\mathbb{S}$ cannot cross ambients of group $\mathbb{S}$. □

Correctness of the Adapted 0CFA Analysis. The correctness of the analysis for Discretionary Ambients is still a "subject reduction" result saying that the validity of an analysis estimate is preserved during execution:

Theorem 25. *If* $(\mathcal{I}, \mathcal{D}) \models^{\star}_{\Gamma} P$ *and* $\Gamma \vdash P \rightarrow^{*} Q$ *then* $(\mathcal{I}, \mathcal{D}) \models^{\star}_{\Gamma} Q$.

Implementation of the Adapted 0CFA Analysis. The Moore Family property still ensures that all processes admit a least analysis estimate:

Theorem 26. *The set* $\{(\mathcal{I}, \mathcal{D}) \mid (\mathcal{I}, \mathcal{D}) \models^{\star}_{\Gamma} P\}$ *is a Moore family.*

Also the implementation in ALFP proceeds exactly as in Section 2.2.

Precision of the Adapted 0CFA Analysis. The 0CFA analysis of Discretionary Ambients seems to be more precise than the corresponding 0CFA analysis of Mobile Ambients. An occurrence of this phenomenon is Example 23 where the analysis of Discretionary Ambients reveals that ambients of group $\mathbb{S}$ will never cross ambients of $\mathbb{S}$; on the other hand the analysis of the Mobile Ambients version of the same process in Example 15 is not able to give the same precise result.

The better precision can be ascribed to the extra information that co-capabilities give about the behaviour of the process. This extra information allows for

additional constraints on the analysis result, which in turn makes the analysis more precise. For example, accumulated errors where one "incorrect element" in the solution gives rise to several more incorrect elements are less likely to occur because it is improbable for an incorrect element to fulfil the extra constraints inferred by the co-capabilities.

Example 27. The analysis gains precision by the way access control is added to the processes. Consider for example the process:

$$\mathsf{a}[\,] \mid \mathsf{b}[\,] \mid \mathsf{c}[\mathsf{b}[\mathsf{in}\ \mathsf{a}]]$$

which is analysed in a type environment where $\Gamma(\mathsf{a}) = \mathbb{A}$, $\Gamma(\mathsf{b}) = \mathbb{B}$, and $\Gamma(\mathsf{c}) = \mathbb{C}$. When analysed with the 0CFA analysis for Mobile Ambients we get the correct but imprecise result $\mathcal{I}_1$ indicating that that b may turn up inside a.

$$\begin{array}{lll}
\mathcal{I}_1(\star) = \{\mathbb{A},\mathbb{B},\mathbb{C}\} & \mathcal{I}_2(\star) = \{\mathbb{A},\mathbb{B},\mathbb{C}\} & \mathcal{I}_3(\star) = \{\mathbb{A},\mathbb{B},\mathbb{C}\} \\
\mathcal{I}_1(\mathbb{A}) = \{\mathbb{B}\} & \mathcal{I}_2(\mathbb{A}) = \emptyset & \mathcal{I}_3(\mathbb{A}) = \{\overline{\mathsf{in}}_{\mathbb{B}}\mathbb{A},\mathbb{B}\} \\
\mathcal{I}_1(\mathbb{B}) = \{\mathsf{in}\ \mathbb{A}\} & \mathcal{I}_2(\mathbb{B}) = \{\mathsf{in}\ \mathbb{A}\} & \mathcal{I}_3(\mathbb{B}) = \{\mathsf{in}\ \mathbb{A}\} \\
\mathcal{I}_1(\mathbb{C}) = \{\mathbb{B}\} & \mathcal{I}_2(\mathbb{C}) = \{\mathbb{B}\} & \mathcal{I}_3(\mathbb{C}) = \{\mathbb{B}\}
\end{array}$$

Suppose that we add access rights to the above process in order *not* to allow b to enter a. In that case, we do not add any co-capabilities and the process above is just viewed as a Discretionary Ambient process. The analysis result $\mathcal{I}_2$ found using the 0CFA analysis of Discretionary Ambients, however, now correctly shows that b cannot show up inside a.

Suppose on the other hand that we add access rights in order to *allow* b to enter a. Then we add the co-capability $\overline{\mathsf{in}}_{\mathbb{B}}\mathsf{a}$ and get the process:

$$\mathsf{a}[\overline{\mathsf{in}}_{\mathbb{B}}\mathsf{a}] \mid \mathsf{b}[\,] \mid \mathsf{c}[\mathsf{b}[\mathsf{in}\ \mathsf{a}]]$$

Now, we get the analysis result $\mathcal{I}_3$ that imprecisely shows that b can turn up in a. We conclude that the additional precision strongly depends on how access rights are added. □

3.3 A Context-Sensitive 1CFA Analysis

In preparation for the study of mandatory access control in the next section we shall now develop a more precise analysis of Discretionary Ambients. Instead of merely recording the *father-son* relationship it takes the grandfather into account and directly records the *grandfather-father-son* relationship by means of a ternary relation.

As before the analysis approximates the behaviour of a process by a *single* abstract configuration that describes all the possible derivatives that the process may have. It distinguishes between the various groups of ambients but not between the individual ambients. Unlike before the analysis is context-sensitive in keeping track of the grandfather relevant for the father-son relationship. Hence the analysis represents the tree structure of the processes by a ternary relation

$$\mathcal{I} : \mathbf{Group} \times \mathbf{Group} \to \mathcal{P}(\mathbf{Group} \cup \mathbf{Cap} \cup \overline{\mathbf{Cap}})$$

so that $\mu_s \in \mathcal{I}\langle \mu_g, \mu_f \rangle$ means that μ_s is a son of μ_f while at the same time μ_f is a son of μ_g. In a similar way the "observation predicate" becomes a ternary relation

$$\mathcal{D} : \mathbf{Group} \times \mathbf{Group} \to \mathcal{P}(\mathbf{Cap} \cup \overline{\mathbf{Cap}})$$

Example 28. For the running example of Example 21 we may use the following definition of $\mathcal{I}$. Here $\top$ is the father of $\star$, i.e. $\top$ is the grandfather of the top-level ambients in the process being analysed. The entries specify the set of sons with a given combination of grandfather and father:

father \ grandfather	$\top$	$\star$	$\mathbb{S}$	$\mathbb{P}$
$\star$	$\{\mathbb{P}, \mathbb{S}\}$			
$\mathbb{S}$		$\{\mathbb{P}, \mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}, \mathsf{open}\ \mathbb{P}, \overline{\mathsf{in}}_{\mathbb{P}}\mathbb{S}, \overline{\mathsf{out}}_{\mathbb{P}}\mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\mathbb{P}\}$		
$\mathbb{P}$		$\{\mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\mathbb{P}\}$	$\{\mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\mathbb{P}\}$	

To be more specific, the fragment $\mathsf{p}[\mathsf{out\ A.in\ B}.\overline{\mathsf{open}}_{\mathbb{S}}\ \mathsf{p}]$ will give rise to $\mathsf{in}\ \mathbb{S} \in \mathcal{I}(\mathbb{S}, \mathbb{P})$, $\mathsf{out}\ \mathbb{S} \in \mathcal{I}(\mathbb{S}, \mathbb{P})$, $\overline{\mathsf{open}}_{\mathbb{S}}\mathbb{P} \in \mathcal{I}(\mathbb{S}, \mathbb{P})$ as shown above. We shall come back to $\mathcal{D}$ in Example 29. □

Specification of the 1CFA Analysis. The judgement of the analysis takes the form

$$(\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} P$$

and expresses that $\mathcal{I}$ and $\mathcal{D}$ are safe approximations of the configurations that P may evolve into when ambient names are mapped to groups as specified by Γ and when $\top$ and $\star$ are the ambient groups of the grandfather and father, respectively.

Analysis of Composite Processes. It is rather straightforward to adapt the clauses for analysing composite processes:

$$\begin{array}{lll}
(\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} (\nu n : \mu)P & \text{iff} & (\mathcal{I}, \mathcal{D}) \models_{\Gamma[n \mapsto \mu]}^{\langle \top, \star \rangle} P \\
(\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} (\nu \mu)P & \text{iff} & (\mathcal{I}, \mathcal{D}) \models_{\Gamma[\mu \mapsto \diamond]}^{\langle \top, \star \rangle} P \\
(\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} \mathbf{0} & \text{iff} & \text{true} \\
(\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} P_1 \mid P_2 & \text{iff} & (\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} P_1 \ \wedge\ (\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} P_2 \\
(\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} !P & \text{iff} & (\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} P \\
(\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} n[P] & \text{iff} & \mu \in \mathcal{I}\langle \top, \star \rangle \ \wedge\ (\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \star, \mu \rangle} P \\
 & & \text{where } \mu = \Gamma(n)
\end{array}$$

The only modification worth observing is the change of ambience in the final rule: the father $\star$ now becomes the grandfather and the ambient μ now becomes the father when analysing the process P.

Analysis of Co-capabilities hardly requires any changes:

$$(\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} \overline{\mathsf{in}}_{\mu} n.\, P \;\text{ iff }\; \overline{\mathsf{in}}_{\mu}\mu' \in \mathcal{I}\langle\top,\star\rangle \wedge (\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} P \quad \text{where } \mu' = \Gamma(n)$$

$$(\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} \overline{\mathsf{out}}_{\mu} n.\, P \;\text{ iff }\; \overline{\mathsf{out}}_{\mu}\mu' \in \mathcal{I}\langle\top,\star\rangle \wedge (\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} P \quad \text{where } \mu' = \Gamma(n)$$

$$(\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} \overline{\mathsf{open}}_{\mu} n.\, P \;\text{ iff }\; \overline{\mathsf{open}}_{\mu}\mu' \in \mathcal{I}\langle\top,\star\rangle \wedge (\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} P \quad \text{where } \mu' = \Gamma(n)$$

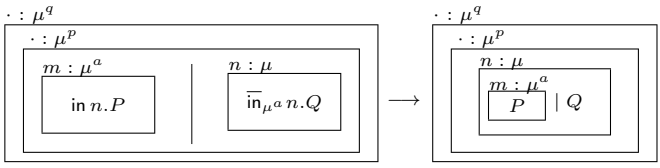
Analysis of Capabilities require a number of changes; we begin with the in-capability (explained below):

$$(\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} \mathsf{in}\ n.\, P \text{ iff}$$

$$\left[\begin{array}{l} \mathsf{in}\ \mu \in \mathcal{I}\langle\top,\star\rangle \wedge (\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} P \wedge \\ \forall \mu^a,\mu^p,\mu^q : \left[\begin{array}{l} \mathsf{in}\ \mu \in \mathcal{I}\langle\mu^p,\mu^a\rangle \wedge \\ \mu^a \in \mathcal{I}\langle\mu^q,\mu^p\rangle \wedge \\ \mu \in \mathcal{I}\langle\mu^q,\mu^p\rangle \wedge \\ \overline{\mathsf{in}}_{\mu^a}\mu \in \mathcal{I}\langle\mu^p,\mu\rangle \end{array}\right] \Rightarrow \left[\begin{array}{l} \mu^a \in \mathcal{I}\langle\mu^p,\mu\rangle \wedge \\ \mathcal{I}\langle\mu^p,\mu^a\rangle \subseteq \mathcal{I}\langle\mu,\mu^a\rangle \wedge \\ \mathsf{in}\ \mu \in \mathcal{D}\langle\mu^p,\mu^a\rangle \wedge \\ \overline{\mathsf{in}}_{\mu^a}\mu \in \mathcal{D}\langle\mu^p,\mu\rangle \end{array}\right] \end{array}\right]$$

where $\mu = \Gamma(n)$

Recall that the semantic rule is:



As before the first step is to identify a potential redex:

- $\mathsf{in}\ \mu \in \mathcal{I}\langle\mu^p,\mu^a\rangle$ ensures that the $\mathsf{in}\ n$ capability is present inside some ambient m; here n has group μ and m has group μ^a whereas μ^p is the group of m's father.
- $\mu^a \in \mathcal{I}\langle\mu^q,\mu^p\rangle$ establishes more of m's (i.e. μ^a's) context: its father is μ^p and its grandfather is μ^q.
- $\mu \in \mathcal{I}\langle\mu^q,\mu^p\rangle$ will now ensure that n (i.e. μ) is a sibling of m: it has the same father μ^p and grandfather μ^q.
- $\overline{\mathsf{in}}_{\mu^a}\mu \in \mathcal{I}\langle\mu^p,\mu\rangle$ finally ensures that n (i.e. μ) grants the access right to m (i.e. μ^a) in the context established by the father μ^p of μ.

In addition to identifying possible n's and m's of the semantic rule these steps also identify the context of the redex and thereby rule out some of the confusion that is inevitable when the ambient names are replaced by groups.

Having identified a potential redex in this way the next step is to record in $\mathcal{I}$ the effect of reducing the redex. This is expressed by:

- $\mu^a \in \mathcal{I}\langle \mu^p, \mu \rangle$ records that m (i.e. μ^a) is moved into n (i.e. μ) and this happens only when μ^p is the father.
- $\mathcal{I}\langle \mu^p, \mu^a \rangle \subseteq \mathcal{I}\langle \mu, \mu^a \rangle$ records that everything inside μ^a with grandfather μ^p (the processes P and Q in the semantic rule) as a result of the reduction also may have grandfather μ.

Note that the latter step is considerably more involved than in the 0CFA analysis due to the need to update the context of all entities moved. Finally we need to update the "observation predicate" $\mathcal{D}$:

- $\mathsf{in}\ \mu \in \mathcal{D}\langle \mu^p, \mu^a \rangle$ records that the in-capability was executed.
- $\overline{\mathsf{in}}_{\mu^a} \mu \in \mathcal{D}\langle \mu^p, \mu \rangle$ records that the $\overline{\mathsf{in}}$-co-capability was executed.

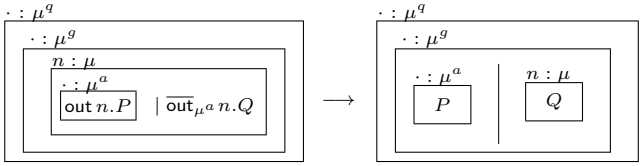
The out-capability follows much the same pattern

$$(\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} \mathsf{out}\ n.\, P \text{ iff}$$

$$\begin{bmatrix} \mathsf{out}(\mu) \in \mathcal{I}\langle \top, \star \rangle \wedge (\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} P \wedge \\ \forall\, \mu^a, \mu^g, \mu^q : \begin{bmatrix} \mathsf{out}(\mu) \in \mathcal{I}\langle \mu, \mu^a \rangle \wedge \\ \mu^a \in \mathcal{I}\langle \mu^g, \mu \rangle \wedge \\ \mu \in \mathcal{I}\langle \mu^q, \mu^g \rangle \wedge \\ \overline{\mathsf{out}}_{\mu^a} \mu \in \mathcal{I}\langle \mu^g, \mu \rangle \end{bmatrix} \Rightarrow \begin{bmatrix} \mu^a \in \mathcal{I}\langle \mu^q, \mu^g \rangle \wedge \\ \mathcal{I}\langle \mu, \mu^a \rangle \subseteq \mathcal{I}\langle \mu^g, \mu^a \rangle \wedge \\ \mathsf{out}(\mu) \in \mathcal{D}\langle \mu, \mu^a \rangle \wedge \\ \overline{\mathsf{out}}_{\mu^a} \mu \in \mathcal{D}\langle \mu^g, \mu \rangle \end{bmatrix} \end{bmatrix}$$

where $\mu = \Gamma(n)$

corresponding to the semantics:



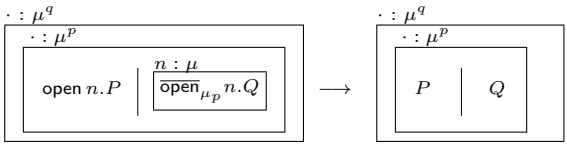
Also the open-capability follows much the same pattern

$$(\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} \mathsf{open}\ n.\, P \text{ iff}$$

$$\begin{bmatrix} \mathsf{open}\ \mu \in \mathcal{I}\langle \top, \star \rangle \wedge (\mathcal{I}, \mathcal{D}) \models_{\Gamma}^{\langle \top, \star \rangle} P \wedge \\ \forall\, \mu^p, \mu^q : \begin{bmatrix} \mathsf{open}\ \mu \in \mathcal{I}\langle \mu^q, \mu^p \rangle \wedge \\ \mu \in \mathcal{I}\langle \mu^q, \mu^p \rangle \wedge \\ \overline{\mathsf{open}}_{\mu^p} \mu \in \mathcal{I}\langle \mu^p, \mu \rangle \end{bmatrix} \Rightarrow \begin{bmatrix} \mathcal{I}\langle \mu^p, \mu \rangle \subseteq \mathcal{I}\langle \mu^q, \mu^p \rangle \wedge \\ \mathsf{open}\ \mu \in \mathcal{D}\langle \mu^q, \mu^p \rangle \wedge \\ \overline{\mathsf{open}}_{\mu^p} \mu \in \mathcal{D}\langle \mu^p, \mu \rangle \end{bmatrix} \end{bmatrix}$$

where $\mu = \Gamma(n)$

corresponding to the semantics:



Example 29. Consider the analysis of our running example of Example 21 (again taking $\Gamma(\mathsf{A}) = \Gamma(\mathsf{B}) = \mathbb{S}$ and $\Gamma(\mathsf{p}) = \mathbb{P}$):

$$(\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} \mathsf{A}\,[\,\mathsf{p}\,[\,\mathsf{out}\,\mathsf{A}.\ \mathsf{in}\,\mathsf{B}.\ \overline{\mathsf{open}}_{\mathbb{S}}\ \mathsf{p}\,]\ \mid\ \overline{\mathsf{out}}_{\mathbb{P}}\ \mathsf{A}\,]\ \mid\ \mathsf{B}\,[\,\overline{\mathsf{in}}_{\mathbb{P}}\ \mathsf{B}.\ \mathsf{open}\,\mathsf{p}\,]$$

This formula will be satisfied when $\mathcal{I}$ is (as in Example 28)

grandfather

father: $\mathcal{I}$	$\top$	$\star$	$\mathbb{S}$	$\mathbb{P}$
$\star$	$\{\mathbb{P},\mathbb{S}\}$			
$\mathbb{S}$		$\{\mathbb{P}, \mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}, \mathsf{open}\ \mathbb{P}, \overline{\mathsf{in}}_{\mathbb{P}}\mathbb{S}, \overline{\mathsf{out}}_{\mathbb{P}}\mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\mathbb{P}\}$		
$\mathbb{P}$		$\{\mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\mathbb{P}\}$	$\{\mathsf{in}\ \mathbb{S}, \mathsf{out}\ \mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\mathbb{P}\}$	

and $\mathcal{D}$ is given by:

grandfather

father: $\mathcal{D}$	$\top$	$\star$	$\mathbb{S}$	$\mathbb{P}$
$\star$				
$\mathbb{S}$		$\{\mathsf{in}\ \mathbb{S}, \mathsf{open}\ \mathbb{P}, \overline{\mathsf{in}}_{\mathbb{P}}\mathbb{S}, \overline{\mathsf{out}}_{\mathbb{P}}\mathbb{S}\}$		
$\mathbb{P}$		$\{\mathsf{in}\ \mathbb{S}\}$	$\{\mathsf{out}\ \mathbb{S}, \overline{\mathsf{open}}_{\mathbb{S}}\mathbb{P}\}$	

One may observe that $\mathcal{D}(\cdots)$ is often a *strict* subset of $\mathcal{I}(\cdots) \cap (\mathbf{Cap} \cup \overline{\mathbf{Cap}})$. This shows that a number of capabilities will never occur in a setting where they are allowed to execute. In particular, even though $\mathcal{I}(\star,\mathbb{P})$ contains $\mathsf{out}\ \mathbb{S}$ as well as $\overline{\mathsf{open}}_{\mathbb{S}}\mathbb{P}$, they are absent in $\mathcal{D}(\star,\mathbb{P})$ and hence cannot execute. □

Correctness of the 1CFA Analysis. The semantic correctness of the analysis is expressed by the following subject reduction result:

Theorem 30. *If* $(\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} P$ *and* $\Gamma \vdash P \to^* Q$ *then* $(\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} Q$.

As before the proof is by induction in the length of the derivation sequence; each step is by induction on the inference in the semantics and uses that structurally congruent processes admit the same analysis results.

Implementation of the 1CFA Analysis. The Moore family property ensures that all processes can be analysed and admit a least analysis estimate:

Theorem 31. *The set* $\{(\mathcal{I},\mathcal{D}) \mid (\mathcal{I},\mathcal{D}) \models_{\Gamma}^{\langle\top,\star\rangle} P\}$ *is a Moore family.*

Though the analysis is more complex that the 0CFA analysis it is still expressible in ALFP using the encodings described in Section 2.2. Hence, the implementation is again done using the Succinct Solver. However, the time complexity of the calculation of the analysis result for the 1CFA analysis is higher than that of the 0CFA analysis although still within polynomial time. We report in Table 5

Table 5. Running times for 0CFA versus 1CFA on four scalable test processes.

	A	B	C	D
0CFA	$O(N^{1.03})$	$O(N^{1.03})$	$O(N^{2.32})$	$O(N^{1.22})$
1CFA	$O(N^{1.98})$	$O(N^{2.00})$	$O(N^{3.34})$	$O(N^{2.01})$
Size of **Group**	$O(N^1)$	$O(N^1)$	$O(N^{1/2})$	$O(N^{2/3})$

on some practical experiments where the time spent for computing the analysis result is expressed in terms of the size N of the process for four scalable test processes. The test processes describe a packet being routed though a network of sites with different network topology.

One reason for the higher complexity of the 1CFA is that the size of the analysis estimate for the 1CFA potentially is larger than size of the analysis estimate for the 0CFA by a factor corresponding to the number of groups in **Group** (i.e. the potential number of elements in $\mathbf{Group} \times \mathbf{Group} \to \mathcal{P}(\mathbf{Group} \cup \mathbf{Cap} \cup \overline{\mathbf{Cap}})$ versus $\mathbf{Group} \to \mathcal{P}(\mathbf{Group} \cup \mathbf{Cap} \cup \overline{\mathbf{Cap}})$). The results for the first two test processes, A and B behave exactly as expected. This is also largely the case for the last test process, D, taking into account that the number of groups here is not linear in the size of the process. The odd-one-out is the result for the test process C; here we conjecture that the 1CFA analysis is more costly than expected because the lower number of groups means that many ambients get mixed up, i.e. that many more contexts have occurrences of the same group. Thus the precision of the 1CFA is outweighed by the imprecision of the group information.

Precision of the 1CFA Analysis. The 1CFA analysis is more precise than the 0CFA analysis in that it records more of the *context* in which capabilities can be used.

Example 32. Recall the process of Example 27

$$\mathsf{a}[\overline{\mathsf{in}}_{\mathbb{B}}\mathsf{a}] \mid \mathsf{b}[\,] \mid \mathsf{c}[\mathsf{b}[\mathsf{in}\ \mathsf{a}]]$$

where the 0CFA of Discretionary Ambients imprecisely finds that b may enter a (as shown by $\mathcal{I}_3$ of Example 27).

The 1CFA analysis gives rise the least estimate $\mathcal{I}$:

$\mathcal{I}$	$\top$	$\star$	$\mathbb{A}$	$\mathbb{B}$	$\mathbb{C}$
$\star$	$\{\mathbb{A}, \mathbb{B}, \mathbb{C}\}$				
$\mathbb{A}$		$\{\overline{\mathsf{in}}_{\mathbb{B}}\mathbb{A}\}$			
$\mathbb{B}$					$\{\mathsf{in}\ \mathbb{A}\}$
$\mathbb{C}$		$\{\mathbb{B}\}$			

It shows that the 1CFA analysis is able to record that the capability in a is not inside b in a context where b is actually a sibling to a. Thus, the analysis result show that b will not show up in a. □

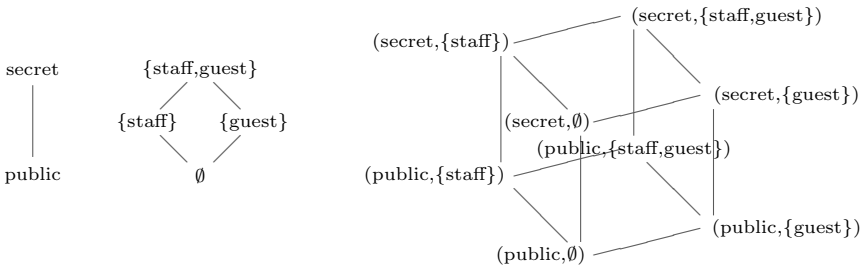


Fig. 5. Example security lattices for confidentiality.

4 Mandatory Access Control

The aim of this section is to show how the Bell-LaPadula [3,22] and Biba [4,22] models can be reformulated for Discretionary Ambients and thereby to construct the appropriate *reference monitor semantics*. The first design decision is to assign security levels/integrity levels to the groups rather than the ambients; an ambient then inherits the level of its group. We shall therefore extend the syntax of group introduction to have the form $(\nu\mu^{\ell})P$ meaning that μ has the level ℓ. It is now straightforward to extend the semantics to map groups to security/integrity levels; the key rule is:

$$\frac{\Gamma[\mu \mapsto \ell] \vdash P \to Q}{\Gamma \vdash (\nu\mu^{\ell})P \to (\nu\mu^{\ell})Q}$$

The security/integrity level information is then used in formalising reference monitors in the spirit of the Bell-LaPadula and Biba models; this is covered in the following subsections and takes the form of defining augmented semantics with judgements of the forms $\Gamma \vdash P \twoheadrightarrow Q$. We shall allow to write $\twoheadrightarrow_{\mathsf{BLP}}$ and $\twoheadrightarrow_{\mathsf{Biba}}$ to differentiate between the two choices.

4.1 Confidentiality: The Bell-LaPadula Model

Dynamic Formulation of the Bell-LaPadula Model. The Bell-LaPadula security model [3,22] is expressed using an access control matrix and an assignment of security levels to objects and subjects; the security levels are arranged in a lattice $(L, \leq)$ where $\ell_1 \leq \ell_2$ means that ℓ_1 has a lower security level than ℓ_2. The overall aim is then to enforce confidentiality by *preventing information from flowing downwards* from a high security level to a low security level.

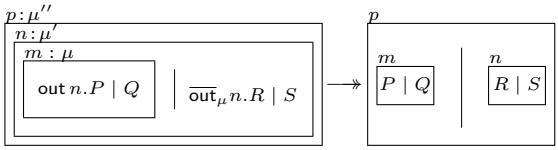
To exemplify our interpretation of the Bell-LaPadula model for ambients we use a simple lattice $(L, \leq)$ with $L = \{\mathsf{public}, \mathsf{secret}\}$ and $\mathsf{public} \leq \mathsf{secret}$ as is one of the possibilities illustrated in Figure 5. Conceptually we regard a *secret ambient* as a protective boundary from which no information is allowed escape outwards to a *public ambience*. Thus, "anything" is allowed to happen inside or outside the boundary but restrictions are imposed on which ambients can leave it. This can be effectuated by making a number of restrictions on when operations (i.e. in, out, and open) on ambients are allowed. Informally, we state these restrictions as follows:

– any ambient can *enter* any other ambient;
– an ambient can only *leave* a secret ambient in a secret ambience; and
– a secret ambient can only *be dissolved* in a secret ambience.

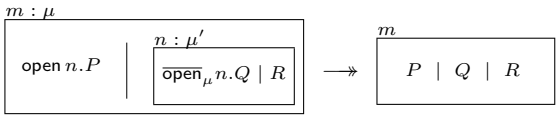
The first item reflects that since nothing moves outwards when an in-capability is executed then confidentiality cannot be effected. This is in contrast to the situation for an out-capability where we must prevent movement from a secrets ambient out to a public ambience. Correspondingly, the third condition expresses that secret information inside a secret ambient is not allowed to flow into a public ambience when the secret ambient is opened.

These conditions can be formalised as side conditions to the semantic rules as shown below. The side conditions are modifications of the previous rules in that they incorporate the dynamic checks to be performed by the reference monitor. As an example, the rule for out now contains information about the encapsulating ambient in order to formalise the second condition. The formulation below is specialised to the security lattice {public, secret} whereas the formulation in Table 6 is sufficiently general to deal with an arbitrary security lattice.

any ambient can enter any other ambient (hence, there is no side condition).



a secret ambient can leave an ambient in a secret ambience:
$\Gamma(\mu) = \mathsf{secret} \Rightarrow \Gamma(\mu'') = \mathsf{secret}$



a secret ambient can be dissolved in a secret ambience:
$\Gamma(\mu') = \mathsf{secret} \Rightarrow \Gamma(\mu) = \mathsf{secret}$

Note that the clause for the out-capability compares the security levels of p and m — and that p and m have a grandfather-son relationship; this is the key reason for why the 1CFA analysis is going to produce better results than the 0CFA analysis.

Example 33. Returning to the running example (expressed in Discretionary Ambients) of Example 21 we first assume that the sites are secret, that the packet is public, and that the overall system is in a public ambience. The packet can move out of the site A because it is public and thus it does not impose the additional conditions on the ambience. The packet can always move into the site B and since it is public it can be opened inside B. So the execution explained in Example 21 is accepted by the reference monitor. This means that the transitions outlined in Example 21 hold for $\rightarrow$ as well as $\twoheadrightarrow$.

Alternatively, let us assume that the sites are public but that the packet is secret. Now the packet is not allowed to leave A unless the overall system is in

Table 6. Reference monitor semantics for Bell-LaPadula.

$$\frac{\Gamma[\mu \mapsto \ell] \vdash P \twoheadrightarrow Q}{\Gamma \vdash (\nu\mu^{\ell})P \twoheadrightarrow (\nu\mu^{\ell})Q} \qquad \frac{\Gamma[n \mapsto \mu] \vdash P \twoheadrightarrow Q}{\Gamma \vdash (\nu n : \mu)P \twoheadrightarrow (\nu n : \mu)Q} \quad \text{if } \mu \in \mathrm{dom}(\Gamma)$$

$$\frac{\Gamma \vdash P \to Q}{\Gamma \vdash n[P] \to n[Q]} \qquad \frac{\Gamma \vdash P \to Q}{\Gamma \vdash P \mid R \to Q \mid R} \qquad \frac{P \equiv P' \quad \Gamma \vdash P' \to Q' \quad Q' \equiv Q}{\Gamma \vdash P \to Q}$$

$$\Gamma \vdash m[\mathsf{in}\ n.\, P \mid Q] \mid n[\overline{\mathsf{in}}_{\mu} n.\, R \mid S] \twoheadrightarrow n[m[P \mid Q] \mid R \mid S] \quad \text{if } \Gamma(m) = \mu$$

$$\Gamma \vdash p[n[m[\mathsf{out}\ n.\, P \mid Q] \mid \overline{\mathsf{out}}_{\mu} n.\, R \mid S]] \twoheadrightarrow p[m[P \mid Q] \mid n[R \mid S]] \quad \text{if } \Gamma(m) = \mu \wedge \Gamma(\mu) \leq \Gamma(\Gamma(p))$$

$$\Gamma \vdash m[\mathsf{open}\ n.\, P \mid n[\overline{\mathsf{open}}_{\mu} n.\, Q \mid R]] \twoheadrightarrow m[P \mid Q \mid R] \quad \text{if } \Gamma(m) = \mu \wedge \Gamma(\Gamma(n)) \leq \Gamma(\mu)$$

a secret ambience. The packet can always move into B but then it cannot be opened because it is secret and the ambience provided by B is indeed public. Thus in this case the reference monitor will "kick in" and prevent the execution from happening. This means that the transitions outlined in Example 21 hold for $\to$ but not for $\twoheadrightarrow$. □

Static Formulation of the Bell-LaPadula Model. Having obtained an approximation to the behaviour of the processes the next step is to formulate the Bell-LaPadula conditions as checks on the analysis results — the idea being that if the analysis result passes these checks then the reference monitor will not intervene in the execution of the process.

The analysis result $(\mathcal{I}, \mathcal{D})$ satisfies the Bell-LaPadula security conditions with respect to the assignment Γ of security levels to groups, written $\mathsf{BLP}_{\Gamma}(\mathcal{I}, \mathcal{D})$, if the following conditions are fulfilled:

$$\forall \mu^a, \mu^g : \left[\exists \mu : \overline{\mathsf{out}}_{\mu^a} \mu \in \mathcal{D}\langle \mu^g, \mu \rangle\right] \Rightarrow \Gamma(\mu^a) \leq \Gamma(\mu^g)$$

$$\forall \mu^p, \mu : \left[\overline{\mathsf{open}}_{\mu^p} \mu \in \mathcal{D}\langle \mu^p, \mu \rangle\right] \Rightarrow \Gamma(\mu) \leq \Gamma(\mu^p)$$

The precondition of the first formula identifies a potential out-redex and the conclusion then requires that the security level of the subject (μ^a) is less than that of the ambience (μ^g) — exactly as required by the reference monitor. The second formula expresses the corresponding condition for the open-redex. In both cases we make good use of the "observation predicate" $\mathcal{D}$ in order to avoid copying large parts of the clauses in the 1CFA analysis.

Example 34. Corresponding to Example 33 let us assume that $\Gamma(\mathbb{S}) = \mathsf{secret}$ and $\Gamma(\mathbb{P}) = \Gamma(\star) = \Gamma(\top) = \mathsf{public}$. We shall check the Bell-LaPadula conditions imposed on the analysis result presented in Example 29. The precondition of the out-capability is only satisfied for $\mu = \mathbb{S}$, $\mu^a = \mathbb{P}$, and $\mu^g = \star$ and the check $\Gamma(\mu^a) \leq \Gamma(\mu^g)$ amounts to $\mathsf{public} \leq \mathsf{public}$, which clearly holds. The

precondition for the open-capability is only satisfied for $\mu = \mathbb{P}$ and $\mu^p = \mathbb{S}$ and the check $\Gamma(\mu) \leq \Gamma(\mu^p)$ amounts to public $\leq$ secret, which also holds. Consequently the analysis result ensures that the reference monitor will not "kick in" and, therefore, its tests can be dispensed with — as was also observed in Example 33.

Alternatively, we may assume that $\Gamma(\mathbb{P}) =$ secret and $\Gamma(\mathbb{S}) = \Gamma(\star) = \Gamma(\top) =$ public. For the out-capability the check $\Gamma(\mu^a) \leq \Gamma(\mu^g)$ amounts to secret $\leq$ public and since this does not hold we cannot guarantee that the reference monitor will not "kick in" — as we reasoned in Example 33. □

The correctness of the static test can be expressed as follows; the result says that we can dispense with the reference monitor if the static checks are fulfilled:

Theorem 35. *Suppose* $\mathsf{BLP}_\Gamma(\mathcal{I}, \mathcal{D})$ *holds for some analysis estimate that satisfies* $(\mathcal{I}, \mathcal{D}) \models_\Gamma^{\langle \top, \star \rangle} P$*; then any execution* $\Gamma \vdash P \rightarrow^* Q$ *can be mimicked as an execution* $\Gamma \vdash P \twoheadrightarrow^*_{\mathsf{BLP}} Q$*.*

The proof is by induction in the length of the derivation sequence using the subject reduction theorem; each step is by induction on the inference in the semantics; we omit the details.

Efficient implementation of the 1CFA analysis is is as before. It is straightforward to translate $\mathsf{BLP}_\Gamma(\mathcal{I}, \mathcal{D})$ into ALFP and the test can be performed in low polynomial time.

4.2 Integrity: The Biba Model

Dynamic Formulation of the Biba Model. The Biba model for integrity [4,22] combines the access control matrix with an assignment of integrity levels to subjects and objects; the integrity levels are arranged in a lattice and the overall aim is to *prevent the corruption of high-level entities by low-level entities*:

As a simple example we use the lattice $\{$dubious, trusted$\}$ with dubious $\leq$ trusted. Again we view ambients as protective boundaries but now we want to prevent *dubious ambients* from moving into *trusted ambients*. We can state this as the following requirements to operations on ambients:

- only trusted ambients can *enter* trusted ambients;
- only trusted ambients can *leave* in a trusted ambience;
- inside a trusted ambient it is only possible to *dissolve* trusted ambients that only contain trusted sub-ambients.

The first item reflects that a trusted ambient will be corrupted if it is entered by a dubious sibling. The second item reflects that a trusted ambient will also be corrupted if it is "entered" by a dubious grandchild. The third item again protects a trusted ambient against corruption by dubious grandchildren but this time as a result of a child being opened. Furthermore, we disallow dubious

Table 7. Reference monitor semantics for Biba.

$$\frac{\Gamma[\mu \mapsto \ell] \vdash P \twoheadrightarrow Q}{\Gamma \vdash (\nu\mu^{\ell})P \twoheadrightarrow (\nu\mu^{\ell})Q} \qquad \frac{\Gamma[n \mapsto \mu] \vdash P \twoheadrightarrow Q}{\Gamma \vdash (\nu n : \mu)P \twoheadrightarrow (\nu n : \mu)Q} \quad \text{if } \mu \in \mathrm{dom}(\Gamma)$$

$$\frac{\Gamma \vdash P \to Q}{\Gamma \vdash n[P] \to n[Q]} \qquad \frac{\Gamma \vdash P \to Q}{\Gamma \vdash P \mid R \to Q \mid R} \qquad \frac{P \equiv P' \quad \Gamma \vdash P' \to Q' \quad Q' \equiv Q}{\Gamma \vdash P \to Q}$$

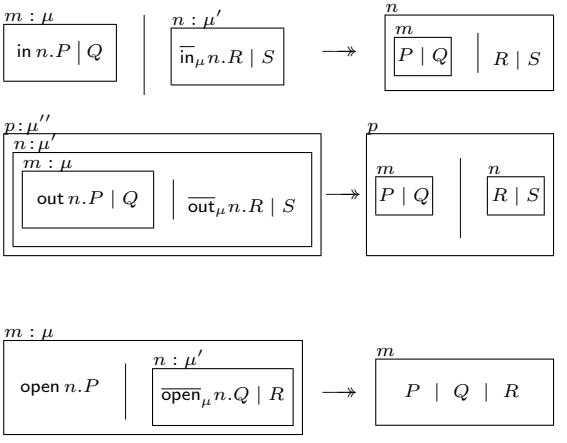
$$\Gamma \vdash m[\mathsf{in}\ n.\, P \mid Q] \mid n[\overline{\mathsf{in}}_{\mu} n.\, R \mid S] \twoheadrightarrow n[m[P \mid Q] \mid R \mid S] \quad \text{if } \Gamma(m) = \mu \wedge \Gamma(\Gamma(n)) \leq \Gamma(\mu)$$

$$\Gamma \vdash p[n[m[\mathsf{out}\ n.\, P \mid Q] \mid \overline{\mathsf{out}}_{\mu} n.\, R \mid S]] \twoheadrightarrow p[m[P \mid Q] \mid n[R \mid S]] \quad \text{if } \Gamma(m) = \mu \wedge \Gamma(\Gamma(p)) \leq \Gamma(\mu)$$

$$\Gamma \vdash m[\mathsf{open}\ n.\, P \mid n[\overline{\mathsf{open}}_{\mu} n.\, Q \mid R]] \twoheadrightarrow m[P \mid Q \mid R] \quad \text{if } \Gamma(m) = \mu \wedge \mu \leq \Gamma(n) \wedge \forall p[\cdot] \in \mathrm{top}(Q \mid R) : \Gamma(\mu) \leq \Gamma(\Gamma(p))$$

ambients to be opened in a trusted ambience, since this could unleash "dubious capabilities".

This is formalised by the following extension of the semantics. As before the formulation below is adapted to the security lattice $\{\mathsf{dubious}, \mathsf{trusted}\}$ whereas the formulation in Table 7 is sufficiently general to deal with an arbitrary security lattice. In the clause for open we write $\mathrm{top}(Q \mid R)$ for the ambients occurring at the *top-level* of the process $Q \mid R$; we demand that all of these ambients must have an integrity level that is at least as high as that of the encapsulating ambient.



only trusted ambients can enter a trusted ambient:
$\Gamma(\mu') = \mathsf{trusted} \Rightarrow \Gamma(\mu) = \mathsf{trusted}$

a trusted ambience can only be entered by a trusted ambient:
$\Gamma(\mu'') = \mathsf{trusted} \Rightarrow \Gamma(\mu) = \mathsf{trusted}$

only trusted ambients with trusted sub-ambients can be dissolved inside a trusted ambient:
$\Gamma(\mu) = \mathsf{trusted} \Rightarrow \Gamma(\mu') = \mathsf{trusted} \wedge \forall p[\cdot] \in \mathrm{top}(Q \mid R) : \Gamma(\Gamma(p)) = \mathsf{trusted}$

Note that also here the ambients of interest have a grandfather-son relationship; once again this motivates our study of the 1CFA analysis.

Example 36. Returning to Example 21 we now assume that sites are dubious, that the packet is trusted, and that the overall system is trusted. The packet can move out of A because the overall system is trusted and it can now move into B because the sites are dubious. Since the site is dubious there is no problem opening the packet although it is trusted. So the execution of Example 21 will be accepted by the reference monitor. This means that the transitions outlined in Example 21 hold for $\rightarrow$ as well as $\twoheadrightarrow$.

Alternatively, assume that the sites are trusted but that the packet as well as the overall system are dubious. Then the reference monitor will prevent the packet from entering the site B as it will corrupt its integrity. This means that the transitions outlined in Example 21 hold for $\rightarrow$ but not for $\twoheadrightarrow$. □

Static Formulation of the Biba Model. The analysis result $(\mathcal{I}, \mathcal{D})$ satisfies the Biba integrity condition with respect to the assignment Γ of integrity levels to groups, written $\mathsf{Biba}_\Gamma(\mathcal{I}, \mathcal{D})$, if the following conditions are fulfilled:

$$\forall \mu, \mu^a : \left[\exists \mu^p : \overline{\mathsf{in}}_{\mu^a}\mu \in \mathcal{D}\langle \mu^p, \mu\rangle\right] \Rightarrow \Gamma(\mu) \leq \Gamma(\mu^a)$$

$$\forall \mu^a, \mu^g : \left[\exists \mu : \overline{\mathsf{out}}_{\mu^a}\mu \in \mathcal{D}\langle \mu^g, \mu\rangle\right] \Rightarrow \Gamma(\mu^g) \leq \Gamma(\mu^a)$$

$$\forall \mu^p, \mu : \ [\overline{\mathsf{open}}_{\mu^p}\mu \in \mathcal{D}\langle \mu^p, \mu\rangle \Rightarrow \\ [\Gamma(\mu^p) \leq \Gamma(\mu) \wedge \forall \mu^c : \mu^c \in \mathcal{I}(\mu^p, \mu) \Rightarrow \Gamma(\mu^p) \leq \Gamma(\mu^c)]]$$

Again the preconditions express the presence of a potential redex and the conclusion then imposes the relevant integrity constraint of the reference monitor. Note that the top-level ambients (μ^c) occurring inside the subject (μ) easily can be accessed using the relation $\mathcal{I}$.

Example 37. Corresponding to Example 36 let us assume that $\Gamma(\mathbb{S}) = \mathsf{dubious}$ and $\Gamma(\mathbb{P}) = \Gamma(\star) = \Gamma(\top) = \mathsf{trusted}$ and let us check the Biba conditions on the analysis result of Example 29. The precondition for the in-capability is only satisfied for $\mu = \mathbb{S}$, $\mu^a = \mathbb{P}$, and $\mu^p = \star$ and the check $\Gamma(\mu) \leq \Gamma(\mu^a)$ amounts to $\mathsf{dubious} \leq \mathsf{trusted}$, which clearly holds. The precondition for the out-capability is only satisfied for $\mu^a = \mathbb{P}$, $\mu = \mathbb{S}$, and $\mu^g = \star$ and the check $\Gamma(\mu^g) \leq \Gamma(\mu^a)$ amounts to $\mathsf{trusted} \leq \mathsf{trusted}$, which also holds. The precondition for the open-capability only holds for $\mu = \mathbb{P}$ and $\mu^p = \mathbb{S}$. This leads to the two requirements that $\mathsf{dubious} \leq \mathsf{trusted}$ and that the universally quantified implication must hold for these values of μ and μ^p. The latter trivially holds since there exists no μ^c to fulfil the precondition reflecting that $\mathbb{P}$ never contains any sub-ambients. Consequently the analysis result ensures that the reference monitor will never "kick in" as we already observed in Example 36.

Alternatively we may assume that $\Gamma(\mathbb{S}) = \mathsf{trusted}$ but $\Gamma(\mathbb{P}) = \Gamma(\star) = \Gamma(\top) = \mathsf{dubious}$. For the in-capability the check $\Gamma(\mu) \leq \Gamma(\mu^a)$ amounts to $\mathsf{trusted} \leq \mathsf{dubious}$ and since this does not hold we cannot guarantee that the reference monitor will not "kick in" — as we already observed in Example 36. □

The correctness of the static test can be expressed as follows; the result says that we can dispense with the reference monitor if the static checks are fulfilled:

Theorem 38. *Suppose* $\mathsf{Biba}_\Gamma(\mathcal{I},\mathcal{D})$ *holds for some analysis estimate that satisfies* $(\mathcal{I},\mathcal{D}) \models_\Gamma^{\langle\top,\star\rangle} P$*; then any execution* $\Gamma \vdash P \rightarrow^* Q$ *can be mimicked as an execution* $\Gamma \vdash P \twoheadrightarrow^*_{\mathsf{Biba}} Q$*.*

The proof is by induction on the length of the derivation sequence using the subject reduction theorem; each step is by induction on the inference in the semantics; we omit the details.

Efficient implementation is as before: translating $\mathsf{Biba}_\Gamma(\mathcal{I},\mathcal{D})$ into ALFP, the test can be performed in low polynomial time.

Remark 39. The static tests for Bell-LaPadula and Biba could also be phrased using the father-son relations found by the 0CFA. Since the tests are of a grandfather-child nature the 0CFA is, however, likely to be too imprecise. Another approach to improving the simple father-son analysis is considered by Braghin, Cortesi, and Focardi in [6] (for the classical Mobile Ambients with labels). Their idea is to extend the analysis with a third component holding the *security level* of the grandfather. While their analysis will in general be more precise than using our 0CFA it is coarser than the one developed here using the 1CFA because the security information of a grandfather may identify a rather large superset of the set of grandfathers possible.

Related work of studying mandatory access control within ambient calculi has also been done by Bugliesi, Castanga, and Crafa [9]. They interpret access control in an ambient setting as access to *communication* between ambients rather than *mobility* as we do. This leads to the definition of a new calculus called Boxed Ambients (see Section 5), which extends the communication primitives of the original Mobile Ambient calculus. Their main result is a type system which checks that communication does not violate a given access policy. The type system primarily builds on the *exchange types* of [14] (see Remark 50) and, as such, is quite far from our analysis which tracks mobility.

More recently, the same authors have studied *information flow* in a variant of Boxed Ambients [21]. They partition information (i.e. names and capabilities) into high and low security levels and define a type system which impose *access control* on low level processes. Next, they define processes to be contextually equivalent whenever they exhibit a certain kind of communication out of *low level* ambients. Finally, they show that a well-typed low level process is equivalent to itself composed with any well-typed high level process. Thus, a low level process cannot observe the difference between running on its own or running together with a well-typed high level process. Thereby they ensure that a low level process cannot *deduce* anything about well-typed high level processes. As the authors themselves point out, the requirement for high level processes to be well-typed is rather strict since it is not ensured that every process can be typed. Therefore, the high-level processes must be known so the method only applies to closed systems. □

5 Cryptographic Protocols

Mobile Ambients as originally introduced in [13] also admit communication primitives. However, rather than following full-fledged channel-based communication as in the π-calculus the designers opted for a more limited form of communication where each ambient has a mailbox that allows both ambient names as well as capabilities to be communicated. In this way all communication is local between the top-level processes of an ambient and one achieves long distance communication by a combination of local communication and movement within the ambient hierarchy. Also they opted for asynchronous rather than synchronous communication. In the case of monadic input and output the asynchronous[2] communication primitives are:

$\langle M \rangle$ outputs the message M asynchronously to the local mailbox;
$(x).P$ inputs a message from the mailbox, binds it to x and continues as P.

Boxed Ambients [9,8] takes the view that ambients should not only be allowed to communicate locally but also with their children (but not grandchildren) and parents (but not grandparents). In the monadic calculus the new communication primitives are

- $\langle M \rangle^{\uparrow}$ outputs a message to the mailbox of the parent;
- $\langle M \rangle^{\circ}$ outputs a message to the local mailbox;
- $\langle M \rangle^{n}$ outputs a message to the mailbox of a child named n;
- $(x)^{\uparrow}.P$ inputs a message from the mailbox of the parent;
- $(x)^{\circ}.P$ inputs a message from the local mailbox;
- $(x)^{n}.P$ inputs a message from the mailbox of a child named n.

Example 40. Boxed Ambients are well suited for expressing *perfect symmetric* cryptography although there is no explicit cryptographic primitives. We shall code symmetric keys as names and introduce them using restriction; the perfect nature of the cryptography is then due to the semantics of restriction, $(\nu n : \mu)P$, that ensures that the name n introduced is distinct from all other names whether already introduced or yet to be introduced. In this model even a brute force attack cannot succeed.

A plain-text message msg encrypted under a key K is then coded as

$$\mathsf{K}[\langle \mathsf{msg} \rangle^{\circ}]$$

whereas decrypting a ciphertext cph under the key K is coded as:

$$\mathsf{cph} \mid (\mathsf{x})^{\mathsf{K}}.\cdots \mathsf{x} \cdots$$

Here the decryption only succeeds if indeed cph contains a top-level ambient of the form $\mathsf{K}[\langle \mathsf{msg} \rangle^{\circ}]$. If the encrypted message needs to survive for later decryption it can be "protected" from destruction by placing a replication operator (!) in front of it. □

[2] To obtain synchronous communication we should write $\langle M \rangle.P$ and modify the semantics.

5.1 Syntax and Semantics of Boxed Ambients

For definition of the syntax of Boxed Ambients we revert to Mobile Ambients as explained in Section 2 and add polyadic communication.

Syntax. As in other presentations of ambients we make a distinction between names (introduced by restrictions) and variables (introduced by input); in our view, this distinction adds clarity both to the semantics and to the analysis. We shall therefore find it helpful to introduce a new syntactic category N of namings that can be both variables and names and to use namings where names where used before. Furthermore we introduce an auxiliary syntactic category η for the communication direction. The syntax then reads:

$$\begin{aligned} P ::= {} & (\nu\mu)P \mid (\nu n : \mu)P \mid \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid N[P] \mid M.\,P \mid \\ & \langle M_1, \cdots, M_k \rangle^\eta \mid (x_1, \cdots, x_k)^\eta.\,P \\ M ::= {} & \mathsf{in}\ N \mid \mathsf{out}\ N \mid N \\ N ::= {} & n \mid x \\ \eta ::= {} & N \mid \uparrow \mid \circ \end{aligned}$$

We follow the designers of Boxed Ambients in *not* including the open-capability. For simplicity of presentation we have not allowed the formation of composite capabilities, i.e., we disallow the nil capability ϵ and the concatenation $M_1.M_2$ as would be needed for communicating complete routes along which movement could take place; most of the development would work for these extensions as well but the actual implementation would be more complex.

Semantics. The semantic changes are rather minor with respect to the specification of Tables 1 and 2. For the structural congruence we simply need to add the rule:

$$P \equiv Q \ \Rightarrow\ (x_1, \cdots, x_k)^\eta.\,P \equiv (x_1, \cdots, x_k)^\eta.\,Q$$

For the transition relation of Table 2 we add a number of rules for communication. The rules are depicted in their monadic version below, i.e. where only one message is communicated at a time. The polyadic version of the rules are summarised in Table 8 where $P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\}$ denotes P with M_i substituted for x_i with the usual α-renaming in order to avoid capturing free names in the M_i. We shall only apply the transition relation to closed processes, i.e. processes without any free variables, and hence M_i contains no variables. Consequently we shall dispense with α-renaming of variables (since this simplifies the specification of the 0CFA analysis).

First we have local communication, which takes place between any two sibling processes

$$\langle M \rangle^\circ \mid (x)^\circ.\,P \quad \longrightarrow \quad P\{x \leftarrow M\}$$

and binds the value of the message M to the variable x in the receiving process P. Next we add the following rules for output to a child, either by explicitly naming the child (and using the child's mailbox for the exchange of the message)

Table 8. Transition relation for Boxed Ambients. Additions to Table 2.

$\langle M_1,\cdots,M_k\rangle^\circ \mid (x_1,\cdots,x_k)^\circ.P$	$\rightarrow$	$P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\}$
$\langle M_1,\cdots,M_k\rangle^n \mid n\,[(x_1,\cdots,x_k)^\circ.P \mid Q]$	$\rightarrow$	$n\,[P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\} \mid Q]$
$\langle M_1,\cdots,M_k\rangle^\circ \mid n\,[(x_1,\cdots,x_k)^\uparrow.P \mid Q]$	$\rightarrow$	$n\,[P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\} \mid Q]$
$(x_1,\cdots,x_k)^\circ.P \mid n\,[\langle M_1,\cdots,M_k\rangle^\uparrow \mid R]$	$\rightarrow$	$P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\} \mid n\,[R]$
$(x_1,\cdots,x_k)^n.P \mid n\,[\langle M_1,\cdots,M_k\rangle^\circ \mid R]$	$\rightarrow$	$P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\} \mid n\,[R]$

$$\langle M\rangle^n \quad \Big| \quad \overset{n}{\boxed{(x)^\circ.P \mid Q}} \quad \longrightarrow \quad \overset{n}{\boxed{P\{x \leftarrow M\} \mid Q}}$$

or anonymously (using the enclosing ambients mailbox for the exchange of the message)

$$\langle M\rangle^\circ \quad \Big| \quad \overset{n}{\boxed{(x)^\uparrow.P \mid Q}} \quad \longrightarrow \quad \overset{n}{\boxed{P\{x \leftarrow M\} \mid Q}}$$

Finally, we add the following rules for output to a parent

$$(x)^\circ.P \quad \Big| \quad \overset{n}{\boxed{\langle M\rangle^\uparrow \mid R}} \quad \longrightarrow \quad P\{x \leftarrow M\} \quad \Big| \quad \overset{n}{\boxed{R}}$$

and

$$(x)^n.P \quad \Big| \quad \overset{n}{\boxed{\langle M\rangle^\circ \mid R}} \quad \longrightarrow \quad P\{x \leftarrow M\} \quad \Big| \quad \overset{n}{\boxed{R}}$$

In particular, we *do not* have a rule for output to grandchildren such as:

$$\langle M_1,\cdots,M_k\rangle^n \mid n\,[m\,[(x_1,\cdots,x_k)^\uparrow.P \mid Q] \mid R] \not\rightarrow \cdots$$

Instead communication between grandparent and its grandchildren will have to be forwarded e.g. as done by m below (for monadic output of a message M):

$$\langle M\rangle^m \mid m[(x)^\circ.\langle x\rangle^\circ \mid n[(x)^\uparrow.\cdots x\cdots]]$$

Example 41. Continuing Example 40 we have that

$$\mathsf{K}[\langle \mathsf{msg}\rangle^\circ] \mid (\mathsf{x})^{\mathsf{K}}.\cdots \mathsf{x} \cdots \quad \rightarrow \quad \mathsf{K}[\ \ ] \mid \cdots \mathsf{msg} \cdots$$

showing that after the decryption an empty message, $\mathsf{K}[\,]$, is left behind. □

Example 42. Boxed Ambients allows to code a package moving around on a network where it communicates the name of a new ambient to be created at the destination:

$$\begin{aligned} & \mathsf{A}[\ \mathsf{p}[\ \mathsf{out}\,\mathsf{A}.\ \mathsf{in}\,\mathsf{B}.\ \langle \mathsf{C}\rangle^{\uparrow}\ ]]\ \mid\ \mathsf{B}[\ (\mathsf{x})^{\circ}.\mathsf{x}[\ ]] \\ \rightarrow\ & \mathsf{A}[\ \ ]\ \mid \mathsf{p}[\ \mathsf{in}\,\mathsf{B}.\ \langle \mathsf{C}\rangle^{\uparrow}\ ]\ \mid\ \mathsf{B}[\ (\mathsf{x})^{\circ}.\mathsf{x}[\ ]] \\ \rightarrow\ & \mathsf{A}[\ \ ]\ \mid\ \mathsf{B}[\ \mathsf{p}[\ \langle \mathsf{C}\rangle^{\uparrow}\ ]\ \mid\ (\mathsf{x})^{\circ}.\mathsf{x}[\ ]] \\ \rightarrow\ & \mathsf{A}[\ \ ]\ \mid\ \mathsf{B}[\ \mathsf{p}[\ \ ]\ \mid\ \mathsf{C}[\ \ ]\ ] \end{aligned}$$

This example illustrates the usefulness of being able to communicate between adjacent layers and why this reduces (if not obviates) the need for the open-capability. □

5.2 Cryptographic Protocols in Boxed Ambients

Boxed Ambients seems rather well suited for expressing a number of cryptographic protocols. In this subsection we consider a number of protocols that involve a server S and agents (or principals) A and B.

Agents present themselves with their name and frequently there is the need to find the corresponding key. In traditional programming languages one might have an array that is indexed with the name and that produces the key, i.e. the key to be used by the server for encryption or decryption of messages from the agent. In Boxed Ambients it is natural to represent the "array" as a process of the form

$$\mathsf{KeyTable} \quad = \quad \mathsf{n}_1[!\langle \mathsf{K}_1\rangle^{\circ}] \quad \mid \quad \cdots \quad \mid \quad \mathsf{n}_m[!\langle \mathsf{K}_m\rangle^{\circ}]$$

corresponding to the name of the principal n_i being mapped to the key K_i. We use replication to ensure that the "array" can be queried any number of times. Hence, whenever a process performs the action

$$\mathsf{KeyTable} \quad \mid \quad (\mathsf{y}_K)^{\mathsf{x}_n}.\cdots \mathsf{y}_K \cdots$$

it will obtain the key y_K corresponding to the agent x_n.

Occasionally there is a need to test that two random numbers are equal before proceeding. In traditional programming languages one would test the equality of n and m using a conditional. In Mobile Ambients the traditional coding trick is to create an ambient, n[ ] and then let an open-capability, open m, guard the then-branch (ignoring the else-branch). In Boxed Ambients we do not have the open-capability and hence will use communication: we create an ambient, $\mathsf{n}[\langle\rangle^{\circ}]$, that performs a local nullary output and then let an input, $()^{\mathsf{m}}.\cdots$, guard the then-branch. In other words

$$\mathsf{n}[\langle\rangle^{\circ}] \quad \mid \quad ()^{\mathsf{m}}.P$$

will block the execution of P unless n equals m.

Whenever a principal sends a message to another principal it would be natural to encode the message in an anonymous packet (e.g p). Often the message consists of some public names together with a message encrypted by some key

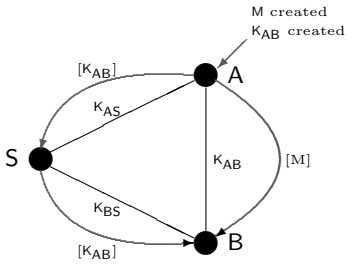


Fig. 6. Wide Mouthed Frog protocol.

(e.g. K) and consisting of some secret names. One could then send the pair (public, secret) from A to B by means of the packet:

$$\mathsf{p}[\mathsf{out\ A.in\ B.}(\langle \mathsf{public} \rangle^{\circ} \mid \mathsf{K}[\mathsf{out\ p.}\langle \mathsf{secret} \rangle^{\circ}])]$$

To avoid an overly heavy coding we shall generally prefer to dispense with the anonymous packet and instead reuse the cryptographic key. We therefore send (public, secret) from A to B by means of:

$$\mathsf{K}[\mathsf{out\ A.in\ B.}(\langle \mathsf{public} \rangle^{\uparrow} \mid \langle \mathsf{secret} \rangle^{\circ})]$$

Once the capabilities out A and in B have been executed the enclosing ambient will have access to the public parts of K without knowing the key, e.g.

$$\mathsf{K}[\langle \mathsf{public} \rangle^{\uparrow} \mid \langle \mathsf{secret} \rangle^{\circ}] \mid (\mathsf{x})^{\circ}.\,\mathsf{x} \longrightarrow \mathsf{K}[\langle \mathsf{secret} \rangle^{\circ}] \mid \mathsf{public}$$

In order to get hold of the secret parts of the message the enclosing ambient needs knowledge of the key K:

$$\mathsf{K}[\langle \mathsf{public} \rangle^{\uparrow} \mid \langle \mathsf{secret} \rangle^{\circ}] \mid (\mathsf{x})^{\mathsf{K}}.\,\mathsf{x} \longrightarrow \mathsf{K}[\langle \mathsf{public} \rangle^{\uparrow}] \mid \mathsf{secret}$$

Wide Mouthed Frog. We consider here the Wide Mouthed Frog protocol originally described in [10] in the simplified version of [1] where agents A and B are given together with a trusted server S. Also secret master keys are in place between the server and the agents: K_{AS} is known only to A and S, and K_{BS} is known only to B and S. Hence the KeyTable to be used in the server is

$$\mathsf{KeyTable} \quad = \quad \mathsf{A}[!\langle \mathsf{K_{AS}} \rangle^{\circ}] \quad \mid \quad \mathsf{B}[!\langle \mathsf{K_{BS}} \rangle^{\circ}]$$

The purpose of the protocol is first to exchange a secret session key K_{AB} for use between A and B, and then to communicate a secret message M using the session key. The protocol depicted in Figure 6 begins with A creating the session key K_{AB} and the message M. Next A forwards the key to S, encoded with K_{AS}, thereby asking S to forward it to B. The key is forwarded to B, encoded with K_{BS}. At this point B is ready to receive the message M communicated by A, this

time encrypted with the secret session key $\mathsf{K_{AB}}$. The classical way of presenting protocols (see e.g. [18]) is to write a narration where the messages of the protocol are listed in order. For each message the principals involved in the message exchange are given along with the content of the message. For the Wide Mouthed Frog protocol this looks as follows (where we write $\mathsf{K}[\,\mathsf{M}\,]$ for the message M encrypted under the key K):

$$\begin{array}{ll} 1.\ \mathsf{A} \to \mathsf{S}: & \mathsf{A}, \mathsf{K_{AS}}[\,\mathsf{B}, \mathsf{K_{AB}}\,] \\ 2.\ \mathsf{S} \to \mathsf{B}: & \mathsf{K_{BS}}[\,\mathsf{A}, \mathsf{K_{AB}}\,] \\ 3.\ \mathsf{A} \to \mathsf{B}: & \mathsf{K_{AB}}[\,\mathsf{M}\,] \end{array}$$

In more detail the steps are as follows (writing Alice for A, Bob for B and Server for S as is customary when discussing protocols):

1. Alice generates a new random session key ($\mathsf{K_{AB}}$)

   $$(\nu\, \mathsf{K_{AB}} : \mathbb{K}_{\mathsf{AB}})$$

   and then sends her name (A), Bobs name (B) and the session key ($\mathsf{K_{AB}}$) to the Server (S) encrypted by her master key $\mathsf{K_{AS}}$:

   $$\mathsf{K_{AS}}[\mathsf{out\ A}.\ \mathsf{in\ S}.\ (\langle \mathsf{A} \rangle^{\uparrow} \mid \langle \mathsf{B}, \mathsf{K_{AB}} \rangle^{\circ})\ ]$$

   As discussed above, Alice's name (A) can be received by any enclosing ambient once $\mathsf{out\ A}.\ \mathsf{in\ S}$ has been executed whereas Bob's name and the session key (B, $\mathsf{K_{AB}}$) can only be received by those ambients holding the master key $\mathsf{K_{AS}}$ under which the message is encrypted.

2. The Server first receives Alice's name ($\mathsf{y_A} = \mathsf{A}$) and obtains her master key ($\mathsf{y}_{\mathsf{K_{AS}}} = \mathsf{K_{AS}}$) from the $\mathsf{KeyTable}$ and uses it to decrypt the remaining components ($\mathsf{y_B} = \mathsf{B}$ and $\mathsf{y}_{\mathsf{K_{AB}}} = \mathsf{K_{AB}}$) of the message:

   $$\mathsf{KeyTable} \mid (\mathsf{y_A})^{\circ}.\, (\mathsf{y}_{\mathsf{K_{AS}}})^{\mathsf{y_A}}.\, (\mathsf{y_B}, \mathsf{y}_{\mathsf{K_{AB}}})^{\mathsf{y}_{\mathsf{K_{AS}}}}.$$

   The Server obtains Bob's master key ($\mathsf{y}_{\mathsf{K_{BS}}} = \mathsf{K_{BS}}$) from $\mathsf{KeyTable}$ and uses it to encrypt Alice's name ($\mathsf{y_A} = \mathsf{A}$) and the random key ($\mathsf{y}_{\mathsf{K_{AB}}} = \mathsf{K_{AB}}$) and sends it to Bob ($\mathsf{y_B} = \mathsf{B}$):

   $$(\mathsf{y}_{\mathsf{K_{BS}}})^{\mathsf{y_B}}.\, \mathsf{y}_{\mathsf{K_{BS}}}[\mathsf{out\ S}.\ \mathsf{in\ y_B}.\ \langle \mathsf{y_A}, \mathsf{y}_{\mathsf{K_{AB}}} \rangle^{\circ}]$$

   Bob decrypts the message using his master key $\mathsf{K_{BS}}$ and obtains Alice's name ($\mathsf{z_A} = \mathsf{A}$) and the session key ($\mathsf{z}_{\mathsf{K_{AB}}} = \mathsf{K_{AB}}$):

   $$(\mathsf{z_A}, \mathsf{z}_{\mathsf{K_{AB}}})^{\mathsf{K_{BS}}}.$$

3. Alice creates her message (M) and sends it encrypted with the session key $\mathsf{K_{AB}}$ to Bob

   $$(\nu\, \mathsf{M} : \mathbb{M})\ \ \mathsf{K_{AB}}[\mathsf{out\ A}.\ \mathsf{in\ B}.\ \langle \mathsf{M} \rangle^{\circ}]$$

   which Bob receives and decrypts using the session key ($\mathsf{z}_{\mathsf{K_{AB}}} = \mathsf{K_{AB}}$):

   $$(\mathsf{x})^{\mathsf{z}_{\mathsf{K_{AB}}}}.\, \cdots \mathsf{x} \cdots$$

The overall protocol can be written as follows:

$$
\begin{array}{l}
\mathsf{A}\ [(\nu\, \mathsf{K_{AB}} : \mathbb{K}_{\mathsf{AB}})\ \mathsf{K_{AS}}[\mathsf{out\ A.\ in\ S.}\ (\langle \mathsf{A}\rangle^{\uparrow} \mid \langle \mathsf{B}, \mathsf{K_{AB}}\rangle^{\circ})\ ]\ \mid \\
\qquad\qquad (\nu\, \mathsf{M} : \mathbb{M})\ \mathsf{K_{AB}}[\mathsf{out\ A.\ in\ B.}\ \langle \mathsf{M}\rangle^{\circ}]\ ] \\
\mid \\
\mathsf{S}\ [\,\mathsf{KeyTable} \mid \\
\quad (\mathsf{y_A})^{\circ}.\ (\mathsf{y_{K_{AS}}})^{\mathsf{y_A}}.\ (\mathsf{y_B}, \mathsf{y_{K_{AB}}})^{\mathsf{y_{K_{AS}}}}. \\
\qquad (\mathsf{y_{K_{BS}}})^{\mathsf{y_B}}.\ \mathsf{y_{K_{BS}}}[\mathsf{out\ S.\ in}\ \mathsf{y_B}.\ \langle \mathsf{y_A}, \mathsf{y_{K_{AB}}}\rangle^{\circ}\ ]\ ] \\
\mid \\
\mathsf{B}\ [(\mathsf{z_A}, \mathsf{z_{K_{AB}}})^{\mathsf{K_{BS}}}.\ (\mathsf{x})^{\mathsf{z_{K_{AB}}}}. \cdots \mathsf{x} \cdots\ ]
\end{array}
$$

We refer to the literature for the security properties of the Wide Mouthed Frog protocol. Actually, our encoding is a bit "more secure" than the original protocol. As an example, in step (1) we ensure that no one can listen to neither Alice's name nor Bob's name and the session key until after the package has been delivered. Rather than attempting to weaken the encoding to open up for more attacks we consider this a benefit of performing the encoding in Boxed Ambients.

Yahalom. The Yahalom protocol is described in [10]; its classical narration is as follows:

1. $\mathsf{A} \to \mathsf{B}: \quad \mathsf{A}, \mathsf{R_A}$
2. $\mathsf{B} \to \mathsf{S}: \quad \mathsf{B}, \mathsf{K_{BS}}[\,\mathsf{A}, \mathsf{R_A}, \mathsf{R_B}\,]$
3. $\mathsf{S} \to \mathsf{A}: \quad \mathsf{K_{AS}}[\,\mathsf{B}, \mathsf{K_{AB}}, \mathsf{R_A}, \mathsf{R_B}\,], \mathsf{K_{BS}}[\,\mathsf{A}, \mathsf{K_{AB}}\,]$
4. $\mathsf{A} \to \mathsf{B}: \quad \mathsf{K_{BS}}[\,\mathsf{A}, \mathsf{K_{AB}}\,], \mathsf{K_{AB}}[\,\mathsf{R_B}\,]$
5. $\mathsf{A} \to \mathsf{B}: \quad \mathsf{K_{AB}}[\,\mathsf{M}\,]$

In Boxed Ambients it can be encoded as follows:

1. Alice sends her name and a random number to Bob:

$$(\nu\, \mathsf{R_A} : \mathbb{R})\ \mathsf{p}[\mathsf{out\ A.\ in\ B.}\ \langle \mathsf{A}, \mathsf{R_A}\rangle^{\uparrow}]$$

   Here there is no encryption in the message and we have to revert to the use of an anonymous package.
2. Bob receives Alice's name and random number and generates his own random number; he then encrypts Alice's name, her random number and his own random number with his own master key and sends it to the Server together with his name:

$$(\mathsf{y_A}, \mathsf{y_R})^{\circ}.\ (\nu\, \mathsf{R_B} : \mathbb{R})\ \ \mathsf{K_{BS}}[\mathsf{out\ B.\ in\ S.}\ (\langle \mathsf{B}\rangle^{\uparrow} \mid \langle \mathsf{y_A}, \mathsf{y_R}, \mathsf{R_B}\rangle^{\circ})]$$

3. The Server receives Bob's name and obtains his master key from KeyTable; he then decrypts Alice's name and the two random numbers and obtains Alice's master key and creates a random session key. Then he constructs *two* messages. The first message is encrypted with Alice's master key and is sent to Alice; it contains Bob's name, the session key, Alice's random number and Bob's random number. The other message is sent to Alice too although intended for Bob (and hence may be instructed to go there) and is encrypted with Bob's master key and consists of Alice's name and the session key:

$$\begin{array}{l}\mathsf{KeyTable} \mid \\ (\mathsf{z_B})^{\circ}.\,(\mathsf{z_{K_{BS}}})^{\mathsf{z_B}}.\,(\mathsf{z_A},\mathsf{z_R},\mathsf{z'_R})^{\mathsf{z_{K_{BS}}}}.\,(\mathsf{z_{K_{AS}}})^{\mathsf{z_A}}.\,(\nu\,\mathsf{K}:\mathbb{K}) \\ \qquad\qquad \mathsf{z_{K_{AS}}}[\mathsf{out\,S.\,in\,z_A}.\langle \mathsf{z_B},\mathsf{K},\mathsf{z_R},\mathsf{z'_R}\rangle^{\circ}] \mid \\ \qquad\qquad \mathsf{z_{K_{BS}}}[\mathsf{out\,S.\,in\,z_A}.\,(\mathsf{y_1},\mathsf{y_2})^{\uparrow}.\,\mathsf{y_1}.\,\mathsf{y_2}.\,\langle \mathsf{z_A},\mathsf{K}\rangle^{\circ}]\end{array}$$

4. Alice decrypts the message intended for her and checks that the random number is her own. Then she sends two messages to Bob: one being the message from the Server and the other being Bob's random number encrypted with the session key:

$$\begin{array}{l}(\mathsf{x_B},\mathsf{x_K},\mathsf{x_R},\mathsf{x'_R})^{\mathsf{K_{AS}}}.\,(\mathsf{x_R}[\langle\rangle^{\circ}] \mid ()^{\mathsf{R_A}}.\,(\,\langle \mathsf{out\,A},\ \mathsf{in\,B}\rangle^{\circ}\ \mid \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{x_K}[\mathsf{out\,A.\,in\,B.}\,\langle \mathsf{x'_R}\rangle^{\circ}]))\end{array}$$

Note that we test the equality of $\mathsf{x_R}$ and $\mathsf{R_A}$ as illustrated above; in a similar way we could have decided to test the equality of $\mathsf{x_B}$ and B (but protocol narrations are often a bit unclear about how many tests actually have to be carried out).
Bob decrypts the first message with his master key thereby obtaining the session key and uses it to decrypt the other message and checks that it equals his random number:

$$(\mathsf{y_A},\mathsf{y_K})^{\mathsf{K_{BS}}}.\,(\mathsf{y_R})^{\mathsf{y_K}}.\,(\mathsf{y_R}[\langle\rangle^{\circ}] \mid ()^{\mathsf{R_B}}.\cdots)$$

5. Alice creates her message and sends it encrypted with the session key to Bob

$$(\nu\,\mathsf{M}:\mathbb{M})\ \ \mathsf{x_K}[\mathsf{out\,A.\ in\,B.}\,\langle \mathsf{M}\rangle^{\circ}]$$

which Bob receives and decrypts using the session key:

$$(\mathsf{y_M})^{\mathsf{y_K}}.\cdots \mathsf{y_M}\cdots$$

The protocol may be summarised as follows:

$$\begin{array}{l}\mathsf{A}\ [(\nu\,\mathsf{R_A}:\mathbb{R})\ \mathsf{p}[\mathsf{out\,A.\ in\,B.}\,\langle \mathsf{A},\mathsf{R_A}\rangle^{\uparrow}]\ \mid \\ \qquad (\mathsf{x_B},\mathsf{x_K},\mathsf{x_R},\mathsf{x'_R})^{\mathsf{K_{AS}}}.\,(\mathsf{x_R}[\langle\rangle^{\circ}] \mid \\ \qquad\qquad ()^{\mathsf{R_A}}.\,(\langle \mathsf{out\,A},\ \mathsf{in\,B}\rangle^{\circ}\ \mid \\ \qquad\qquad \mathsf{x_K}[\mathsf{out\,A.\,in\,B.}\,\langle \mathsf{x'_R}\rangle^{\circ}]\ \mid \\ \qquad\qquad (\nu\,\mathsf{M}:\mathbb{M})\ \ \mathsf{x_K}[\mathsf{out\,A.\ in\,B.}\,\langle \mathsf{M}\rangle^{\circ}]))] \\ \mid \\ \mathsf{S}\ [\mathsf{KeyTable} \mid \\ \quad (\mathsf{z_B})^{\circ}.\,(\mathsf{z_{K_{BS}}})^{\mathsf{z_B}}.\,(\mathsf{z_A},\mathsf{z_R},\mathsf{z'_R})^{\mathsf{z_{K_{BS}}}}.\,(\mathsf{z_{K_{AS}}})^{\mathsf{z_A}}.\,(\nu\,\mathsf{K}:\mathbb{K}) \\ \qquad\qquad \mathsf{z_{K_{AS}}}[\mathsf{out\,S.\,in\,z_A}.\langle \mathsf{z_B},\mathsf{K},\mathsf{z_R},\mathsf{z'_R}\rangle^{\circ}]\ \mid \\ \qquad\qquad \mathsf{z_{K_{BS}}}[\mathsf{out\,S.\,in\,z_A}.\,(\mathsf{y_1},\mathsf{y_2})^{\uparrow}.\,\mathsf{y_1}.\,\mathsf{y_2}.\,\langle \mathsf{z_A},\mathsf{K}\rangle^{\circ}]\,] \\ \mid \\ \mathsf{B}\ [(\mathsf{y_A},\mathsf{y_R})^{\circ}.\ (\nu\,\mathsf{R_B}:\mathbb{R})\ \ \mathsf{K_{BS}}[\mathsf{out\,B.\ in\,S.}\,(\langle \mathsf{B}\rangle^{\uparrow} \mid \langle \mathsf{y_A},\mathsf{y_R},\mathsf{R_B}\rangle^{\circ})] \mid \\ \qquad\qquad (\mathsf{y_A},\mathsf{y_K})^{\mathsf{K_{BS}}}.\,(\mathsf{y_R})^{\mathsf{y_K}}.\,(\mathsf{y_R}[\langle\rangle^{\circ}] \mid ()^{\mathsf{R_B}}.\,(\mathsf{y_M})^{\mathsf{y_K}}.\cdots \mathsf{y_M}\cdots)]\end{array}$$

We refer to the literature for the security properties of the Yahalom protocol.

Needham-Schroeder. The Needham-Schroeder symmetric key protocol is described in [27]; its classical protocol narration is as follows

$$\begin{array}{lll} 1. & \mathsf{A} \to \mathsf{S}: & \mathsf{A}, \mathsf{B}, \mathsf{R_A} \\ 2. & \mathsf{S} \to \mathsf{A}: & \mathsf{K_{AS}}[\,\mathsf{R_A}, \mathsf{B}, \mathsf{K_{AB}}, \mathsf{K_{BS}}[\,\mathsf{A}, \mathsf{K_{AB}}\,]\,] \\ 3. & \mathsf{A} \to \mathsf{B}: & \mathsf{K_{BS}}[\,\mathsf{A}, \mathsf{K_{AB}}\,] \\ 4. & \mathsf{B} \to \mathsf{A}: & \mathsf{K_{AB}}[\,\mathsf{R_B}\,] \\ 5. & \mathsf{A} \to \mathsf{B}: & \mathsf{K_{AB}}[\,\mathsf{R_B} - 1\,] \\ 6. & \mathsf{A} \to \mathsf{B}: & \mathsf{K_{AB}}[\,\mathsf{M}\,] \end{array}$$

In Boxed Ambients this can be encoded as follows:

1. Alice sends her name, Bob's name and a random number to the Server:

$$(\nu\, \mathsf{R_A} : \mathbb{R})\ \mathsf{p}[\mathsf{out\ A.\ in\ S.}\ \langle \mathsf{A}, \mathsf{B}, \mathsf{R_A} \rangle^{\uparrow}\,]$$

2. The Server receives the message, generates a random session key K, and sends a single message to Alice encrypted with her master key. It contains Alice's random number and the session key. It also contains a message intended for Bob (and hence may be instructed to go there) and encrypted with Bob's master key containing the session key and Alice's name. To make this work the message sent to Alice must act as a forwarder from the ambience of the message to the message intended for Bob (as discussed previously):

$$\begin{array}{l} \mathsf{KeyTable} \mid (\mathsf{y_A}, \mathsf{y_B}, \mathsf{y_R})^{\circ}.\, (\mathsf{y}_{\mathsf{K_{AS}}})^{\mathsf{y_A}}.\, (\mathsf{y}_{\mathsf{K_{BS}}})^{\mathsf{y_B}}.\, (\nu\, \mathsf{K} : \mathbb{K}) \\ \qquad \mathsf{y}_{\mathsf{K_{AS}}}[\mathsf{out\ S.\ in}\, \mathsf{y_A}.\ (\langle \mathsf{y_R}, \mathsf{K} \rangle^{\circ} \mid \\ \qquad\qquad (\mathsf{y_1}, \mathsf{y_2}, \mathsf{y_3})^{\circ}.\, \langle \mathsf{y_1}, \mathsf{y_2}, \mathsf{y_3} \rangle^{\circ} \mid \\ \qquad\qquad \mathsf{y}_{\mathsf{K_{BS}}}[(\mathsf{y_1}, \mathsf{y_2}, \mathsf{y_3})^{\uparrow}.\ \mathsf{y_1}.\, \mathsf{y_2}.\, \mathsf{y_3}.\, \langle \mathsf{y_A}, \mathsf{K} \rangle^{\circ}])\ ] \end{array}$$

3. Alice decrypts the message and checks that she got her random number back; then she sends the included message to Bob:

$$(\mathsf{x_R}, \mathsf{x_K})^{\mathsf{K_{AS}}}.\, (\mathsf{x_R}[\langle\rangle^{\circ}] \mid ()^{\mathsf{R_A}}.\, \langle \mathsf{out\ K_{AS}},\ \mathsf{out\ A},\ \mathsf{in\ B} \rangle^{\mathsf{K_{AS}}})$$

4. Bob decrypts the message using his master key, generates a random number, encrypts it with the session key and sends it to Alice:

$$(\mathsf{z_A}, \mathsf{z_K})^{\mathsf{K_{BS}}}.\ \ (\nu\, \mathsf{R_B} : \mathbb{R})\, \mathsf{z_K}[\mathsf{out\ B.\ in}\, \mathsf{z_A}.\ \langle \mathsf{R_B} \rangle^{\circ}]$$

5. Alice decrypts the message using the session key, she modifies Bobs random number (by duplicating it rather than subtracting one), encrypts it with the session key and sends it back to Bob:

$$(\mathsf{x'_R})^{\mathsf{x_K}}.\ \ \mathsf{x_K}[\mathsf{out\ A.\ in\ B.}\ \langle \mathsf{x'_R}, \mathsf{x'_R} \rangle^{\circ}]$$

Bob decrypts the message with the session key and verifies that it is obtained from his random number:

$$(\mathsf{z_R}, \mathsf{z_{R'}})^{\mathsf{x_K}}.\, (\mathsf{z_R}[\langle\rangle^{\circ}] \mid \mathsf{z_{R'}}[\langle\rangle^{\circ}] \mid\ ()^{\mathsf{R_B}}.\, ()^{\mathsf{R_B}}.\, \cdots)$$

6. Alice creates her message and sends it encrypted with the session key to Bob

$$(\nu\,\mathsf{M}:\mathbb{M})\ \ \mathsf{x_K}[\mathsf{out\,A.\ in\,B.}\ \langle\mathsf{M}\rangle^\circ]$$

which Bob receives and decrypts using the session key:

$$(\mathsf{z_M})^{\mathsf{z_K}}.\cdots\mathsf{z_M}\cdots$$

The protocol may be summarised as follows:

$$\begin{array}{l}
\mathsf{A}\ [(\nu\,\mathsf{R_A}:\mathbb{R})\ \mathsf{p}[\mathsf{out\,A.\ in\,S.}\ \langle\mathsf{A},\mathsf{B},\mathsf{R_A}\rangle^\uparrow]\ \mid \\
\qquad\qquad (\mathsf{x_R},\mathsf{x_K})^{\mathsf{K_{AS}}}.(\mathsf{x_R}[\langle\rangle^\circ]\mid()^{\mathsf{R_A}}.(\langle\mathsf{out\,K_{AS}},\ \mathsf{out\,A},\ \mathsf{in\,B}\rangle^{\mathsf{K_{AS}}}\ \mid \\
\qquad\qquad\qquad\qquad (\mathsf{x'_R})^{\mathsf{x_K}}.\ \mathsf{x_K}[\mathsf{out\,A.\ in\,B.}\ \langle\mathsf{x'_R},\mathsf{x'_R}\rangle^\circ]\mid \\
\qquad\qquad\qquad\qquad (\nu\,\mathsf{M}:\mathbb{M})\,\mathsf{x_K}[\mathsf{out\,A.\ in\,B.}\ \langle\mathsf{M}\rangle^\circ]\ ))\ ] \\
\mid \\
\mathsf{S}\ [\mathsf{KeyTable}\ \mid\ (\mathsf{y_A},\mathsf{y_B},\mathsf{y_R})^\circ.(\mathsf{y_{K_{AS}}})^{\mathsf{y_A}}.(\mathsf{y_{K_{BS}}})^{\mathsf{y_B}}.(\nu\,\mathsf{K}:\mathbb{K}) \\
\qquad\qquad\qquad\qquad \mathsf{y_{K_{AS}}}[\mathsf{out\,S.\ in}\,\mathsf{y_A}.\ (\langle\mathsf{y_R},\mathsf{K}\rangle^\circ\ \mid \\
\qquad\qquad\qquad\qquad\qquad\qquad (\mathsf{y_1},\mathsf{y_2},\mathsf{y_3})^\circ.\langle\mathsf{y_1},\mathsf{y_2},\mathsf{y_3}\rangle^\circ\mid \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{y_{K_{BS}}}[(\mathsf{y_1},\mathsf{y_2},\mathsf{y_3})^\uparrow.\ \mathsf{y_1}.\mathsf{y_2}.\mathsf{y_3}.\langle\mathsf{y_A},\mathsf{K}\rangle^\circ])\ ]\ ] \\
\mid \\
\mathsf{B}\ [(\mathsf{z_A},\mathsf{z_K})^{\mathsf{K_{BS}}}.\ (\nu\,\mathsf{R_B}:\mathbb{R})\ \mathsf{z_K}[\mathsf{out\,B.\ in}\,\mathsf{z_A}.\ \langle\mathsf{R_B}\rangle^\circ]\ \mid \\
\qquad\qquad\qquad (\mathsf{z_R},\mathsf{z_{R'}})^{\mathsf{x_K}}.(\,\mathsf{z_R}[\langle\rangle^\circ]\mid\mathsf{z_{R'}}[\langle\rangle^\circ]\ \mid \\
\qquad\qquad\qquad\qquad\qquad ()^{\mathsf{R_B}}.()^{\mathsf{R_B}}.(\mathsf{z_M})^{\mathsf{z_K}}.\cdots\mathsf{z_M}\cdots)\ ]
\end{array}$$

We refer to the literature for the security properties of the Needham-Schroeder protocol.

5.3 Adapting the 0CFA Analysis to Deal with Communication

As far as the analysis is concerned we revert to the 0CFA based analysis presented in Section 2.2. As before we need to have a component

$$\mathcal{I} : \mathbf{Group} \to \mathcal{P}(\mathbf{Group} \cup \mathbf{Cap})$$

keeping track of whom is inside whom. Additionally we need a component keeping track of the values bound to variables

$$\mathcal{R} : \mathbf{Var} \to \mathcal{P}(\mathbf{Group} \cup \mathbf{Cap})$$

and a component keeping track of the contents of the mailboxes:

$$\mathcal{C} : \mathbf{Group} \to \mathcal{P}((\mathbf{Group} \cup \mathbf{Cap})^*)$$

Judgements then take the form

$$(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\mu}_{\Gamma} P$$

where there is only one superscript to $\models$ because we are reverting to the context-insensitive 0CFA based analysis.

Example 43. Suppose that an ambient A has gotten hold of a message encrypted under the key K and that the message instructs A where to move. Assuming that A knows the key K this may be illustrated by the process

$$\mathsf{A}\,[\mathsf{K}\,[\langle \mathsf{in}\,\mathsf{S}\rangle^{\circ}] \mid (\mathsf{x})^{\mathsf{K}}.\,\mathsf{x}] \mid \mathsf{S}\,[\,]$$

Assuming that we analyse the process in an environment Γ with $\Gamma(\mathsf{A}) = \mathbb{A}$, $\Gamma(\mathsf{K}) = \mathbb{K}$, and $\Gamma(\mathsf{S}) = \mathbb{S}$ the analysis estimate

$$\begin{array}{ll} \mathcal{I}(\star) = \{\mathbb{A}, \mathbb{S}\} & \mathcal{C}(\mathbb{K}) = \{\mathsf{in}\,\mathbb{S}\} \\ \mathcal{I}(\mathbb{A}) = \{\mathbb{K}, \mathsf{in}\,\mathbb{S}\} & \mathcal{C}(\star) = \mathcal{C}(\mathbb{A}) = \mathcal{C}(\mathbb{S}) = \emptyset \\ \mathcal{I}(\mathbb{S}) = \{\mathbb{A}\} & \\ \mathcal{I}(\mathbb{K}) = \emptyset & \mathcal{R}(\mathsf{x}) = \{\mathsf{in}\,\mathbb{S}\} \end{array}$$

will show the possible behaviour of the process. This estimate is in fact the best estimate found by the analysis specified below. The ambient hierarchy is once again recorded in $\mathcal{I}$ and records that A may turn up inside S (i.e. $\{\mathbb{A}\} \subseteq \mathcal{I}(\mathbb{S})$). Inspection of $\mathcal{C}$ shows that communication may only take place within K where the capability $\mathsf{in}\,\mathsf{S}$ may be communicated. The variable environment $\mathcal{R}$ shows that the variable x may be bound to exactly this value.

Note that while capabilities are recorded directly in $\mathcal{I}$, communication primitives are recorded indirectly. Outputs are recorded by the effect they have on the mailboxes, i.e. on the content of $\mathcal{C}$, while inputs show up as the possible values in $\mathcal{R}$ of the variables that will be bound by the input. □

Specification of the Communication Analysis. As before each acceptable analysis estimate for a composite process must also be an acceptable analysis estimate for its sub-processes:

$$\begin{array}{lll} (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} (\nu\, n\colon \mu)\, P & \text{iff} & (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma[n\mapsto\mu]} P \\ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} (\nu\, \mu)\, P & \text{iff} & (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma[\mu\mapsto\diamond]} P \\ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} \mathbf{0} & \text{iff} & \text{true} \\ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P_1 \mid P_2 & \text{iff} & (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P_1 \ \wedge\ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P_2 \\ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} !P & \text{iff} & (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P \\ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} N\,[P] & \text{iff} & \forall \mu \in \mathcal{N}_{\Gamma,\mathcal{R}}(N) : \mu \in \mathcal{I}(\star) \ \wedge\ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\mu}_{\Gamma} P \end{array}$$

One change from before is that the name of an ambient can also be given by a variable. We therefore use the auxiliary function $\mathcal{N}_{\Gamma,\mathcal{R}}$ to map namings to sets of groups:

$$\begin{array}{l} \mathcal{N}_{\Gamma,\mathcal{R}}(x) = \mathcal{R}(x) \cap \mathbf{Group} \\ \mathcal{N}_{\Gamma,\mathcal{R}}(n) = \{\mu\} \text{ where } \mu = \Gamma(n) \end{array}$$

This function can be extended to obtain the function $\mathcal{M}_{\Gamma,\mathcal{R}}$ for mapping capabilities to sets of values (capabilities *or* names):

$$\begin{aligned}
\mathcal{M}_{\Gamma,\mathcal{R}}(\mathsf{in}\, N) &= \{\mathsf{in}\, \mu \mid \mu \in \mathcal{N}_{\Gamma,\mathcal{R}}(N)\} \\
\mathcal{M}_{\Gamma,\mathcal{R}}(\mathsf{out}\, N) &= \{\mathsf{out}\, \mu \mid \mu \in \mathcal{N}_{\Gamma,\mathcal{R}}(N)\} \\
\mathcal{M}_{\Gamma,\mathcal{R}}(x) &= \mathcal{R}(x) \\
\mathcal{M}_{\Gamma,\mathcal{R}}(n) &= \{\mu\} \text{ where } \mu = \Gamma(n)
\end{aligned}$$

In Boxed Ambients there is no open-capability so we only need to adapt the clauses for the in- and out-capabilities. Since we shall use the function $\mathcal{M}_{\Gamma,\mathcal{R}}$ we have an additional quantification over μ; for in-capabilities we have

$$\begin{aligned}
(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} \mathsf{in}\, N.\ P \text{ iff } \ & \mathcal{M}_{\Gamma,\mathcal{R}}(\mathsf{in}\, N) \subseteq \mathcal{I}(\star) \ \wedge \ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P \ \wedge \\
& \forall\, \mathsf{in}\, \mu \in \mathcal{M}_{\Gamma,\mathcal{R}}(\mathsf{in}\, N) : \varphi_{\mathsf{in}}(\mu)
\end{aligned}$$

where the "closure condition" φ_{in} is defined by

$$\begin{aligned}
\varphi_{\mathsf{in}}(\mu) \text{ iff } \ \forall\, \mu^a, \mu^p : \ & \mathsf{in}\ \mu \in \mathcal{I}(\mu^a) \ \wedge \\
& \mu^a \in \mathcal{I}(\mu^p) \ \wedge \\
& \mu \in \mathcal{I}(\mu^p) \\
& \Rightarrow\ \mu^a \in \mathcal{I}(\mu)
\end{aligned}$$

For out-capabilities we have

$$\begin{aligned}
(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} \mathsf{out}\, N.\ P \text{ iff } \ & \mathcal{M}_{\Gamma,\mathcal{R}}(\mathsf{out}\, N) \subseteq \mathcal{I}(\star) \ \wedge \ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P \ \wedge \\
& \forall\, \mathsf{out}\, \mu \in \mathcal{M}_{\Gamma,\mathcal{R}}(\mathsf{out}\, N) : \varphi_{\mathsf{out}}(\mu)
\end{aligned}$$

where the "closure condition" φ_{out} is defined by

$$\begin{aligned}
\varphi_{\mathsf{out}}(\mu) \text{ iff } \ \forall\, \mu^a, \mu^g : \ & \mathsf{out}\ \mu \in \mathcal{I}(\mu^a) \wedge \\
& \mu^a \in \mathcal{I}(\mu) \ \wedge \\
& \mu \in \mathcal{I}(\mu^g) \\
& \Rightarrow\ \mu^a \in \mathcal{I}(\mu^g)
\end{aligned}$$

Finally, for "mixed capabilities" we have:

$$\begin{aligned}
(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} N.\ P \text{ iff } \ & \mathcal{M}_{\Gamma,\mathcal{R}}(N) \cap \mathbf{Cap} \subseteq \mathcal{I}(\star) \ \wedge \ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P \ \wedge \\
& \forall\, \mathsf{in}\, \mu \in \mathcal{M}_{\Gamma,\mathcal{R}}(N) : \varphi_{\mathsf{in}}(\mu) \ \wedge \\
& \forall\, \mathsf{out}\, \mu \in \mathcal{M}_{\Gamma,\mathcal{R}}(N) : \varphi_{\mathsf{out}}(\mu)
\end{aligned}$$

Thus, if N is a variable containing a capability then the capability must be in $\mathcal{I}(\star)$ and the "closure conditions" ensures that this capability is analysed all ambients where it may end up.

The new clauses deal with polyadic input and output for each of the three directions. For local communication the clauses are

$$\begin{aligned}
(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} \langle M_1, \cdots, M_k \rangle^{\circ} \quad & \text{iff } \ \mathcal{M}_{\Gamma,\mathcal{R}}(M_1) \times \cdots \times \mathcal{M}_{\Gamma,\mathcal{R}}(M_k) \subseteq \mathcal{C}(\star) \\
(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} (x_1, \cdots, x_k)^{\circ}.P \quad & \text{iff } \ \forall (v_1, \cdots, v_k) \in \mathcal{C}(\star) : \\
& \qquad v_1 \in \mathcal{R}(x_1) \wedge \cdots \wedge v_k \in \mathcal{R}(x_k) \ \wedge \\
& \qquad (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P
\end{aligned}$$

where output ensures that the values are "put" into the local mailbox $\mathcal{C}(\star)$ while input "copies" values from $\mathcal{C}(\star)$ into the variables. For communication with a child we add the clauses:

$$(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} \langle M_1,\cdots,M_k\rangle^N \text{ iff } \forall \mu \in \mathcal{N}_{\Gamma,\mathcal{R}}(N) : \mu \in \mathcal{I}(\star) \Rightarrow \mathcal{M}_{\Gamma,\mathcal{R}}(M_1)\times\cdots\times\mathcal{M}_{\Gamma,\mathcal{R}}(M_k) \subseteq \mathcal{C}(\mu)$$

$$(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} (x_1,\cdots,x_k)^N.P \text{ iff } \forall \mu \in \mathcal{N}_{\Gamma,\mathcal{R}}(N) : \mu \in \mathcal{I}(\star) \Rightarrow \forall (v_1,\cdots,v_k) \in \mathcal{C}(\mu) : v_1 \in \mathcal{R}(x_1) \wedge \cdots \wedge v_k \in \mathcal{R}(x_k) \wedge (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P$$

Here we obtain all the possible groups μ of the child N and check that the corresponding ambient indeed occurs in the ambience $\star$; for each successful group μ the communication is recorded by adapting the clause for local communication.

For communication with a parent we add the clauses:

$$(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} \langle M_1,\cdots,M_k\rangle^{\uparrow} \text{ iff } \forall \mu : \star \in \mathcal{I}(\mu) \Rightarrow \mathcal{M}_{\Gamma,\mathcal{R}}(M_1)\times\cdots\times\mathcal{M}_{\Gamma,\mathcal{R}}(M_k) \subseteq \mathcal{C}(\mu)$$

$$(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} (x_1,\cdots,x_k)^{\uparrow}.P \text{ iff } \forall \mu : \star \in \mathcal{I}(\mu) \Rightarrow \forall (v_1,\cdots,v_k) \in \mathcal{C}(\mu) : v_1 \in \mathcal{R}(x_1) \wedge \cdots \wedge v_k \in \mathcal{R}(x_k) \wedge (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P$$

One may note that the analysis is a bit imprecise with respect to the semantics because although $\langle M\rangle^n \mid n\,[m\,[(x)^{\uparrow}.P \mid Q\,] \mid R\,] \not\rightarrow \cdots$ the analysis will pretend that the communication succeeds. One should also point out that the analysis of the communication primitives would have been somewhat more complex if the designers of Boxed Ambients had decided to keep the open-primitive; the reason is that then the communication may take place in other ambiences than where first encountered in the analysis (see [30] for how to deal with this).

Correctness of the Communication Analysis. Once more the correctness of the analysis amounts to a "subject reduction" result:

Theorem 44. *If* $(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P$ *and* $P \rightarrow^* Q$ *then* $(\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} Q$.

Implementation of the Communication Analysis. Once more the existence of a least analysis is a consequence of the Moore family result:

Theorem 45. *For each* P*, the set* $\{(\mathcal{I},\mathcal{C},\mathcal{R}) \mid (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\star}_{\Gamma} P\}$ *is a Moore family.*

The implementation in ALFP proceeds much as before. One caveat is the use of the universal quantification over μ in the clause for ambients:

$$\forall \mu \in \mathcal{N}_{\Gamma,\mathcal{R}}(N) : \ (\mathcal{I},\mathcal{C},\mathcal{R}) \models^{\mu}_{\Gamma} P$$

Here we must make sure that the variable μ is not in the domain or range of Γ.

Another issue is that the communication component $\mathcal{C}$ uses *sequences* to represent messages in the mailboxes. Instead of encoding sequences in general into a relational form that can be described in ALFP we use the fact that the analysis (and the semantics) only compares messages of the same length. Therefore, we can split $\mathcal{C}$ into mappings $\mathcal{C}_k : \mathbf{Group} \to \mathcal{P}((\mathbf{Group} \cup \mathbf{Cap})^k)$ for each message arity k occurring in the process we analysis. This intuitively corresponds to having different mailboxes for messages of different length. We reformulate the analysis into an equivalent analysis where analysing a communication primitive of arity k only gives requirements to the content of the communication component $\mathcal{C}_k$. In turn, this allows us to encode each communication component $\mathcal{C}_k$ as a relation of *fixed* arity $k + 1$.

We then obtain an implementation much as before. However, now it is no longer the case that there is a fixed upper bound on the nesting depth of quantifiers and therefore the worst-case complexity is exponential; in practice it will be polynomial if there is a small nesting depth of ambients in the original process (or at least when the nesting depth of variable-named ambients is less than linear in the size of the process).

If we were to admit composite capabilities (i.e. $M.M$) we would face the problem that the universe is no longer finite for a given process since the process may output a few simple capabilities and then repeatedly input two capabilities and then output their composition. A rather crude way of dealing with this (taken in [30]) is to abandon recording the causal structure of capabilities. The better way is to adapt the solving technology to deal with a possibly infinite universe. A possibly infinite subset of the universe is then described using a tree grammar (or tree automaton). We can express this using our Succinct Solver by manually translating the specification into one that constructs the tree grammar; we refer to [33] for an account of how to do so for the Spi-calculus [1]. Alternatively we may replace the Succinct Solver with a more appropriate solver based on set-constraints [2] or H3 [34].

Remark 46. Our restriction to finitary calculi is in line with some recent developments for Mobile Ambients. In [17] Charatonik, Gordon, and Talbot provide a type system, which can check whether a process has finite behaviour. It is then possible to model check such a process against a so-called ambient logic [15]. Teller, Zimmer, and Hirschkoff [40] models *a resource* as an ambient R with a fixed capacity given by the maximal number of other ambients allowed inside R at top level. They provide a type system for resource control that checks whether the resource capacities in a system may be exceeded at run-time. □

5.4 Protocol and Exchange Analysis

Protocol Analysis. We now show a number of properties of the Wide Mouthed Frog protocol that can be obtained using the 0CFA analysis.

Example 47. We can analyse the Wide Mouthed Frog protocol using the 0CFA analysis and aim at "maximal precision" by keeping as many groups distinct as

possible. This means that we analyse the protocol in a group environment where $\Gamma(\mathsf{A}) = \mathbb{A}$, $\Gamma(\mathsf{B}) = \mathbb{B}$, $\Gamma(\mathsf{S}) = \mathbb{S}$, $\Gamma(\mathsf{K_{AS}}) = \mathbb{K}_{\mathsf{AS}}$ and $\Gamma(\mathsf{K_{BS}}) = \mathbb{K}_{\mathsf{BS}}$. Recall that the protocol additionally specifies the groups of the session key and the message (i.e. $(\nu\, \mathsf{K_{AB}} : \mathbb{K}_{\mathsf{AB}})$ and $(\nu\, \mathsf{M} : \mathbb{M})$). Then the possible values of variables in the analysis component $\mathcal{R}$ are given by:

$\mathcal{R}$	x	$\mathsf{z_{K_{AB}}}$	$\mathsf{z_A}$	$\mathsf{y_{K_{BS}}}$	$\mathsf{y_{K_{AB}}}$	$\mathsf{y_B}$	$\mathsf{y_{K_{AS}}}$	$\mathsf{y_A}$
	$\{\mathbb{M}\}$	$\{\mathbb{K}_{\mathsf{AB}}\}$	$\{\mathbb{A}\}$	$\{\mathbb{K}_{\mathsf{BS}}\}$	$\{\mathbb{K}_{\mathsf{AB}}\}$	$\{\mathbb{B}\}$	$\{\mathbb{A}, \mathbb{K}_{\mathsf{AS}}\}$	$\{\mathbb{A}\}$

Note in particular that $\mathsf{z_{K_{AB}}}$ can only be bound to the session key created by Alice. Consequently Bob may receive the messages Alice sends encrypted under the session key as shown by the values of $\mathcal{R}(\mathsf{x})$. The reason that both $\mathbb{A}$ and $\mathbb{K}_{\mathsf{AS}}$ show up in $\mathcal{R}(\mathsf{y_{K_{AS}}})$ is that the analysis does not distinguish the ambient A that represents Alice and the ambient A in the $\mathsf{KeyTable}$. □

Example 48. In the specification of the Wide Mouthed Frog protocol the server only allows *known principals* to participate in the protocol, since keys shared between the server and a principal needs to appear in $\mathsf{KeyTable}$. Suppose that an unknown principal E (for Enemy) tries to participate in the protocol by carrying out the part of Alice. That is, suppose that the Enemy is as the process describing Alice with A replaced by E everywhere. This is the process below where $\mathsf{K_{ES}}, \mathsf{M}', \mathbb{K}_{\mathsf{EB}}$, and $\mathbb{M}'$ are arbitrarily chosen free names and free groups, respectively.

$$\begin{array}{l}\mathsf{E}\ [(\nu\, \mathsf{K_{EB}} : \mathbb{K}_{\mathsf{EB}})\ \mathsf{K_{ES}}[\mathsf{out\ E.\ in\ S.}\ (\langle \mathsf{E} \rangle^{\uparrow} \mid \langle \mathsf{B}, \mathsf{K_{EB}} \rangle^{\circ})\ ]\ \mid \\ \qquad\qquad (\nu\, \mathsf{M}' : \mathbb{M}')\ \mathsf{K_{EB}}[\mathsf{out\ E.\ in\ B.}\ \langle \mathsf{M}' \rangle^{\circ}]\ ]\end{array}$$

Since the Server does not know the key $\mathsf{K_{ES}}$ the attempt to participate in the protocol will fail. This can in fact be guaranteed using the 0CFA analysis by analysing the process implementing the Wide Mouthed Frog protocol in parallel with the ambient E above. The interesting part of the analysis result is:

$\mathcal{R}$	x	$\mathsf{z_{K_{AB}}}$
	$\{\mathbb{M}\}$	$\{\mathbb{K}_{\mathsf{AB}}\}$

which guarantees that Bob *will never* receive the message M' in the variable x. Since $\mathsf{K_{ES}}$ and M' are arbitrarily chosen the result guaranties that *any* such attempt by the Enemy to conform with the protocol will indeed fail.

Suppose on the other hand that the Server does know the key $\mathsf{K_{ES}}$, i.e. that $\mathsf{KeyTable}$ is extended to:

$$\mathsf{KeyTable}' \quad = \quad \mathsf{A}\ [!\langle \mathsf{K_{AS}} \rangle^{\circ}] \mid \mathsf{B}\ [!\langle \mathsf{K_{BS}} \rangle^{\circ}] \mid \mathsf{E}\ [!\langle \mathsf{K_{ES}} \rangle^{\circ}]$$

Now the Server may successfully let E participate in the protocol as confirmed by the analysis result:

$\mathcal{R}$	x	$\mathsf{z_{K_{AB}}}$
	$\{\mathbb{M}, \mathbb{M}'\}$	$\{\mathbb{K}_{\mathsf{AB}}, \mathbb{K}_{\mathsf{EB}}\}$

This corresponds to the Enemy being a dishonest principal rather than an intruder. □

Example 49. Assume that an enemy E shares a key $\mathsf{K_{ES}}$ with the Server in the Wide Mouthed Frog protocol (i.e. that the Server uses $\mathsf{KeyTable'}$ of Example 48). The enemy may now attempt to "impersonate Alice" by sending an A in the unencrypted part of the first message. This can be encoded as the process

$$\begin{array}{l}\mathsf{E}\ [(\nu\,\mathsf{K_{EB}} : \mathbb{K}_{\mathsf{EB}})\ \mathsf{K_{ES}}[\mathsf{out\,E.\ in\,S.}\ (\langle \mathbf{A}\rangle^{\uparrow} \mid \langle \mathsf{B}, \mathsf{K_{EB}}\rangle^{\circ})\ ]\ \mid \\ \qquad\qquad (\nu\,\mathsf{M'} : \mathbb{M}')\ \mathsf{K_{EB}}[\mathsf{out\,E.\ in\,B.}\ \langle \mathsf{M'}\rangle^{\circ}]\ ]\end{array}$$

The goal of the Enemy is to "fool" Bob into believing that the key $\mathsf{K_{EB}}$ was a session key generated by Alice. However, the 0CFA analysis guarantees us that this never happens. By analysing the above ambient E in parallel with the Wide Mouth Frog protocol (using $\mathsf{KeyTable'}$ instead of $\mathsf{KeyTable}$) we get:

$\mathcal{R}$	x	$\mathsf{z_{K_{AB}}}$
	$\{\mathbb{M}\}$	$\{\mathbb{K}_{\mathsf{AB}}\}$

This in fact ensures that the key $\mathsf{K_{EB}}$ *will never* even reach Bob i.e. it will never be bound to the variable $\mathsf{z_{K_{AB}}}$. Thus, in particular, this attack cannot fool Bob into believing that the key came from Alice. □

Exchange Analysis. Besides opening and crossing control also *exchange analysis* of ambients is considered in [12] (and links back to the type system of [14]). Exchange analysis deals with determining what the topic of conversation is for some ambient n: is there no communication at all, is there an exchange of ambient names only, or is there an exchange of capabilities only.

These questions can be answered using the analysis as follows. Suppose that $(\mathcal{I}, \mathcal{C}, \mathcal{R}) \models^{\star}_{\Gamma} P$ and let $\mu = \Gamma(n)$.

- No communication at all takes place in case $\mathcal{C}(\mu) = \emptyset$;.
- There is exchange of ambient names only, in case $\mathcal{C}(\mu) \subseteq \mathbf{Group}^*$;
- There is exchange of capabilities only, in case $\mathcal{C}(\mu) \subseteq \mathbf{Cap}^*$.

The correctness of these claims are immediate consequences of Theorem 44. We obtain the best answer by using the least analysis estimate as guaranteed by Theorem 45.

Remark 50. Numerous type systems for ambient calculi deals with communication and much of the work is based on the original type system by Cardelli and Gordon [14] including type systems for Boxed Ambients [9,8,21,26]. Furthermore, some type systems incorporates information about mobility (e.g. [11,12]), which is also the case for Merro and Sassone's recent type system [26] for Boxed Ambients.

The type system of [26] checks whether *exchange types* of send and receive are compatible both for local and non-local communication. If any pairs of communications within a process are incompatible the process does not type check. In comparison, our 0CFA analysis will analyse *any* process; as such, our 0CFA analysis is closer to soft typing. The information conveyed by the exchange types

is, in essence, similar to the content of the $\mathcal{C}$ component of our 0CFA analysis as described in the section above. A notable difference is that [26] forbids communications of different arities within the same ambient. This restriction is present in many type systems for ambient calculi and probably goes back to the type system of [14].

Also the type system of [26] incorporates *mobility types* that for a given ambient n gives the set of other ambients in which n is allowed to occur; this is similar to the $\mathcal{I}$ component of our 0CFA analysis. All types are ordered by a subtyping relation that allows to say that some types are "better" than others and to define a best type. Though listed as work-in-progress, they do not provide a *type inference* algorithm so they cannot automatically calculate the mobility and communication behaviour of a given process. □

6 Conclusion

Mobile Ambients and their variants have established themselves as a useful class of process algebras in which to study mobility. Our first aim was to extend the calculus to express *discretionary access control* in a manner compatible with the classical studies of operating systems; we achieved this goal by developing the Discretionary Ambients (and thereby generalising the Safe Ambients). Our second aim was to extend the calculus to express *mandatory access control* for confidentiality as well as integrity; we achieved this goal by modifying the semantics to enforce the checks of the reference monitor. Our third aim was to show that cryptographic key exchange protocols could be coded rather naturally in Boxed Ambients where we make use of the more general communication primitives of Boxed Ambients over those of Mobile Ambients; as far as we are aware this is the first treatment of key exchange protocols in a calculus for mobility.

Throughout we have defined the semantics and developed 0CFA or 1CFA analyses for the calculi studied. They could be implemented in our Succinct Solver by re-expressing the specification in a fragment of Alternation-free Least Fixed Point Logic (ALFP). Except for Boxed Ambients we could guarantee a worst-case complexity being a polynomial of a low degree; for Boxed Ambients the degree of the polynomial is proportional to the nesting depth of ambients in the original process (and hence exponential to the size of the process in the worst case).

We believe that our Flow Logic approach to analysis gives us a number of conveniences. We share with type systems the convenience of separating specification from implementation thereby obtaining a conceptually cleaner formulation of the analysis that interacts well with semantic correctness and state-of-the-art techniques for efficient implementation. But unlike most type systems based on subtyping we achieve polynomial time complexity for most of the analyses of interest.

Perhaps more importantly the logical or constraint-based nature of our approach lead us to the formulation of "hardest attackers": a finite process characterising all possible malicious processes (somewhat in the manner of hard

problems for a given complexity class). The key element in our success is that we limit our attention to the finitary world of the static analysis. In the original development [31,30] we considered the firewall described in [13]. Here only agents knowing the required passwords are supposed to enter, and it is shown that all agents in a special form will in fact enter. However, it is at least as important to ensure that an attacker not knowing the required passwords cannot enter, since this presents a useful technique for screening a system against attackers. This is achieved using the "hardest attacker" [31,30]. We conjecture that a similar development may be possible for the key exchange protocols considered in Subsection 5.2; a preliminary study for protocols expressed in the LySa-calculus is reported in [5].

Acknowledgements

This research was funded in part by the SecSaf project (number 9901590) funded by the Danish Natural Science Research Council and by the DEGAS project (number IST-2001-32072) funded by the European Union and this paper is also published as DEGAS report number WP6-IMM-I9-Int-001.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols – The Spi calculus. *Information and Computation*, 148(1):1–70, 1999.
2. A. Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming (SCP)*, 35(2):79–111, 1999.
3. D. Bell and L. LaPadula. Secure computer system: Unified exposition and Multics interpretation. Technical Report ESDTR-75-306, MTR-2547, MITRE Corporation, 1975.
4. K. J. Biba. Integrity consideration for secure computer systems. Technical Report ESDTR-76-372, MTR-3153, MITRE Corporation, 1977.
5. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Polynomial-time validation of protocol narration. Manuscript, 2002.
6. C. Braghin, A. Cortesi, and R. Focardi. Control flow analysis for information flow security in Mobile Ambients. In *Proceedings of NordSec 2001*. Technical Report IMM-TR-2001-14. Technical University of Denmark, 2001.
7. M. Buchholtz, F. Nielson, and H. Riis Nielson. Experiments with Succinct Solvers. Technical Report IMM-TR-2002-4, Technical University of Denmark, 2002.
8. M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In *Theoretical Aspects in Computer Science (TACS 2001)*, volume 2215 of *Lecture Notes in Computer Science*, pages 37–63. Springer, 2001.
9. M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in Mobile Ambients. In *CONCUR 2001 – Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 102–120. Springer, 2001.
10. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, pages 18–36, 1990.

11. L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for Mobile Ambients. In *Automata, Languages, and Programming. 26th International Colloquium, ICALP 1999*, volume 1644 of *Lecture Notes in Computer Science*, pages 230–239. Springer, 1999.
12. L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *Theoretical Computer Science. International Conference IFIP TCS 2000*, volume 1872 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2000.
13. L. Cardelli and A. D. Gordon. Mobile Ambients. In *Foundations of Software Science and Computation Structures (FoSSaCS 1998)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.
14. L. Cardelli and A. D. Gordon. Types for Mobile Ambients. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1999)*, pages 79–92. ACM Press, 1999.
15. L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for Mobile Ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, pages 365–377. ACM Press, 2000.
16. L. Cardelli and A. D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
17. W. Charatonik, A. D. Gordon, and J.-M. Talbot. Finite-control Mobile Ambients. In *Programming Languages and Systems. 11th European Symposium on Programming (ESOP 2001)*, volume 2305 of *Lecture Notes in Computer Science*. Springer, 2001.
18. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. http://www-users.cs.york.ac.uk/∼jac/papers/drareviewps.ps, 1997.
19. P. Cousot and R. Cousot. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1977)*, pages 238–252. ACM Press, 1977.
20. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1979)*, pages 269–282. ACM Press, 1979.
21. S. Crafa, M. Bugliesi, and G. Castagna. Information flow security for Boxed Ambients. In *F-WAN: Foundations of Wide Area Network Computing*, volume 63 of *Electronic Notes in Theoretical Computer Science*, 2002.
22. D. Gollmann. *Computer Security*. Wiley, 1999.
23. F. Levi and S. Maffeis. An abstract interpretation framework for analysing Mobile Ambients. In *Static Analysis, 8th International Symposium, SAS 2001*, pages 395–411. Springer, 2001.
24. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, pages 352–364. ACM Press, 2000.
25. D. McAllester. On the complexity analysis of static analyses. In *Static Analysis, 6th International Symposium, SAS 1999*, volume 1694 of *Lecture Notes in Computer Science*, pages 312–329. Springer, 1999.
26. M. Merro and V. Sassone. Typing and subtyping mobility in Boxed Ambients. In *CONCUR 2002 – Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 304–320. Springer, 2002.
27. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
28. F. Nielson and H. Riis Nielson. Flow Logics and operational semantics. *Electronic Notes in Theoretical Computer Science*, 10, 1998.

29. F. Nielson, H. Riis Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.
30. F. Nielson, H. Riis Nielson, and R. R. Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283(2):381–418, 2002.
31. F. Nielson, H. Riis Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in Mobile Ambients. In *CONCUR 1999 – Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 463–477. Springer, 1999.
32. F. Nielson, H. Riis Nielson, and H. Seidl. Automatic complexity analysis. In *Programming Languages and Systems. 11th European Symposium on Programming (ESOP 2002)*, number 2305 in Lecture Notes in Computer Science, pages 243–261. Springer, 2002.
33. F. Nielson, H. Riis Nielson, and H. Seidl. Cryptographic analysis in cubic time. *Electronic Notes in Theoretical Computer Science*, 62, 2002.
34. F. Nielson, H. Riis Nielson, and H. Seidl. Normalizable Horn clauses, strongly recognizable relations and Spi. In *Static Analysis, 9th International Symposium, SAS 2002*, volume 2477 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2002.
35. F. Nielson, H. Riis Nielson, and H. Seidl. Succinct Solvers. Manuscript, 2002.
36. F. Nielson and H. Seidl. Control-flow analysis in cubic time. In *Programming Languages and Systems. 10th European Symposium on Programming (ESOP 2001)*, volume 2028 of *Lecture Notes in Computer Science*, pages 252–268. Springer, 2001.
37. H. Riis Nielson and F. Nielson. Shape analysis for Mobile Ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, pages 142–154. ACM Press, 2000.
38. H. Riis Nielson and F. Nielson. Shape analysis for Mobile Ambients. *Nordic Journal of Computing*, 8:233–275, 2001.
39. H. Riis Nielson and F. Nielson. Flow Logic: a multi-paradigmatic approach to static analysis. In *The Essence of Computation: Complexity, Analysis, Transformation*, Lecture Notes in Computer Science. Springer, to appear.
40. D. T. Teller, P. Zimmer, and D. Hirschkoff. Using ambients to control resources. In *CONCUR 2002 – Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

Author Index

Aldini, Alessandro 1

Blundo, Carlo 44
Bravetti, Mario 1
Buchholtz, Mikael 207
Bugliesi, Michele 91

Castagna, Giuseppe 91
Crafa, Silvia 91

D'Arco, Paolo 44

Focardi, Riccardo 91, 139

Gennaro, Rosario 186
Gorrieri, Roberto 1, 139

Hankin, Chris 1

Martinelli, Fabio 139

Nielson, Flemming 207
Nielson, Hanne Riis 207

Pierro, Alessandra Di 1

Sassone, Vladimiro 91

Wiklicky, Herbert 1