USING PROCESS CALCULI TO MODEL AND VERIFY SECURITY PROPERTIES IN REAL LIFE COMMUNICATION PROTOCOLS

ANDRÉS ALBERTO ARISTIZÁBAL PINZÓN
HUGO ANDRÉS LÓPEZ ACOSTA

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
INGENIER DE SISTEMAS Y COMPUTACIÓN
SANTIAGO DE CALI
2005

USING PROCESS CALCULI TO MODEL AND VERIFY SECURITY PROPERTIES IN REAL LIFE COMMUNICATION PROTOCOLS

ANDRÉS ALBERTO ARISTIZÁBAL PINZÓN
HUGO ANDRÉS LÓPEZ ACOSTA

*Tesis de grado para optar al título de
Ingeniero de Sistemas y Computación*

Director
CAMILO RUEDA CALDERÓN
Ingeniero de Sistemas y Computación

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
INGENIER DE SISTEMAS Y COMPUTACIÓN
SANTIAGO DE CALI
2005

Santiago de Cali, January 20, 2006

Doctor
JORGE FRANCISCO ESTELA URIBE
Decano Académico de la Facultad de Ingeniería
Pontificia Universidad Javeriana
Ciudad

Certifico que el presente trabajo de grado, titulado "USING PROCESS CALCULI TO MODEL AND VERIFY SECURITY PROPERTIES IN REAL LIFE COMMUNICATION PROTOCOLS" realizado por ANDRÉS ALBERTO ARISTIZÁBAL PINZÓN y HUGO ANDRÉS LÓPEZ ACOSTA, estudiantes de Ingeniería de Sistemas y Computación, se encuentra terminado y puede ser presentado para sustentación.

Atentamente,

[illegible]

Ing. CAMILO RUEDA CALDERÓN
Director del Proyecto

Santiago de Cali, January 20, 2006

Doctor
JORGE FRANCISCO ESTELA URIBE
Decano Académico de la Facultad de Ingeniería
Pontificia Universidad Javeriana
Ciudad

Por medio de ésta, presentamos a usted el trabajo de grado titulado "USING PROCESS CALCULI TO MODEL AND VERIFY SECURITY PROPERTIES IN REAL LIFE COMMUNICATION PROTOCOLS" para optar el título de Ingeniero de Sistemas y Computación.

Esperamos que este trabajo reúna todos los requisitos académicos y cumpla el propósito para el cual fue creado, y sirva de apoyo para futuros proyectos en la Universidad Javeriana relacionados con la materia.

Atentamente,

[illegible] [illegible]

ANDRÉS ALBERTO ARISTIZÁBAL PINZÓN HUGO ANDRÉS LÓPEZ ACOSTA

Nota de Aceptación:

Aprobado por el comité de Trabajo de Grado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana para optar el título de Ingeniero de Sistemas y Computación.

[illegible]

Dr. JORGE FRANCISCO ESTELA URIBE
Decano Académico de la Facultad de Ingeniería

[illegible]

MARIA CONSTANZA PABÓN
Directora de la Carrera de Ingeniería de Sistemas y Computación

[illegible]

CAMILO RUEDA CALDERÓN
Director de Tesis

Frank D. Valencia

Dr. Frank D. Valencia
Jurado

[illegible]

Carlos A. Olarte
Jurado

Andrés Aristizábal

To God, my parents: Hersilia, Alvaro and my girlfriend Paola, but specially to my mother for her love and unconditional support.

Hugo Andrés López

To the Lord, my parents: Victor Hugo, Amparo and my sister Veronica, who have never disappointed me.

Acknowledgements

Considering the debts accumulated during the accomplishment of this work, we can hardly express our gratitude to all the people who have helped us directly or indirectly:

- First of all, we wish to thank Camilo Rueda, Director of this thesis, for his valuables comments, ideas and help along the development of this work.
- We owe thanks to Frank Valencia who so much inspired and taught us, following all the research in this work. His support and patience were essential for the final achievement of these results.
- We are in debt to Federico Crazzolara, who with his comments and knowledge outlined and corrected primer versions of this work.
- We are grateful to the Avispa Resarch Group, specially to Jorge A. Pérez, Gustavo Gutiérrez and Julián Gutiérrez, with which we interchanged valuable discussions and comments that contributed to the final version of this work. We owe twice to Jorge A. Pérez, whose accurate and detailed comments and revisions of this work were a priceless improving to the document
- We thank the Universidad Javeriana and the Engineering department, who funded this work as part of the research project ”Modelamiento de Problemas de Ciencia y Tecnología Usando Cálculos de Procesos Concurrentes” realized by the AVISPA Research Group between the months of January 2005 and February 2006.
- We owe thanks to the Comète Team LIX École Polytechnique, directed by Catuscia Palamidessi, and specially to one of its members, Tom Chotia, for his important remarks to the work.
- Finally thanking our parents and all our relatives, including Paola Andrea and Diana Carolina, for their patience, understanding and warm support that never failed.
- And to everyone who directly or indirectly helped with the development of this work.

Contents

List of Tables

List of Figures

Resumen

La seguridad puede considerarse como una de las características más importantes en las comunicaciones actuales. La necesidad de transmitir información crítica de manera segura utilizando canales públicos cobra especial importancia en el contexto de sistemas de cómputo globales como Internet. La abrumadora presencia de estos sistemas en la vida cotidiana, hace que garantizar sus propiedades de seguridad sea un verdadero reto dentro de la teoría de la computación.

En este contexto, los métodos formales consituyen una alternativa para el diseño correcto de mecanismos de comunicación segura. Se trata de abstraer los aspectos esenciales de los protocolos de comunicación en términos de especificaciones formales que puedan ser rigurosamente verificadas. De esta forma, implementaciones derivadas de estas especificaciones tienen una sólida garantía de su correcto funcionamiento. Los *cálculos de procesos* son lenguajes formales de especificación, especialmente creados para desarrollar especificaciones abstractas de sistemas concurrentes y móviles. Estos cálculos ofrecen operaciones para la descripción precisa de los sistemas, así como mecanismos para el análisis en el tiempo de las especificaciones desarrolladas. De forma general, este trabajo explora el uso de un cálculo de procesos concurrente en el análisis, diseño y especificación de protocolos de comunicación. En concreto, este trabajo propone SPL como un cálculo de procesos adecuado para la verificacion de propiedades de sistemas Peer-to-Peer (P2P). De esta forma, MUTE y FTN, dos protocolos de comunicacion para este tipo de sistemas son analizados utilizando los elementos formales provistos por SPL. Se trata de esquemas de comunicación ampliamente relevantes en la actualidad: mientras que el primero representa un esquema general para compartir recursos en una red dinámica, el segundo está orientado a la reconfiguración de aplicaciones en ambientes colaborativos. Las propiedades de seguridad más relevantes para cada uno de estos protocolos son identificadas y analizadas. Este estudio se ve complementado con nuevas versiones de los protocolos que corrigen falencias de seguridad. Una contribución adicional consiste en una serie de codificaciones (encodings) que facilitan la descripción de ciertos tipos de protocolos de comunicaciones concurrentes; estos encodings se mantienen conservativos con respecto a los elementos existentes en SPL.

De esta forma, este trabajo presenta resultados positivos en el campo de la verificación formal de protocolos de seguridad utilizando cálculos de procesos. El presente trabajo da fé tanto de la aplicabilidad de estos formalismos en el modelamiento de sistemas de comunicación concurrente de la vida real, como en el hallazgo de falencias de seguridad asociadas a los protocolos estudiados.

Abstract

Security assurance can be seen as one of the most important characteristics in nowadays communication systems. The need of confidential and reliable transfer of critical information using public channels emerges with special importance in contexts where open and mobile networks are crucial for the accurate behavior of distributed tasks, such as wireless networks or the Internet. The overwhelming presence of this kind of systems in our daily life turns out the correct achievement of security warranties into one of the most important challenges in theory of computation.

In this context, formal methods arise as one of the alternatives for the correct design of secure communication mechanisms, focusing in abstracting essential aspects from communication protocols in terms of formal specifications that can be rigorously verified. In this way, implementations derived from these specifications obtain a solid warranty of their correct behavior. Concretely, *process calculi* are a set of formal languages intended for the specification and verification of concurrent and mobile systems, offering primitives well suited for the precise description of these systems, as well as reasoning techniques for the analysis of the specifications acting concurrently over time.

This work explores the use of concurrent process calculi in the analysis, design, specification and verification of communication protocols. In particular, it proposes the use of SPL as a process calculus well suited for the analysis and verification of security properties over Peer-to-Peer(P2P) systems. In this way, MUTE and FTN, two protocols that clearly represent the behavior of distributed communication systems over open networks, are modelled and verified in SPL. The first protocol portrays a general method for sharing resources over a dynamic network and the second is oriented to the dynamic reconfiguration of applications in collaborative environments. Security properties relevant for each of these protocols are identified and analyzed by means of process calculi, bearing witness of the applicability of this kind of reasonings. This work is complemented with modified versions of the protocols, correcting the security holes encountered in previous versions. In addition, a set of encodings are modelled in SPL, easing the description of concurrent protocols specified in other approaches.

1 Introduction

This thesis explores the use of formal models for the analysis and verification of security properties in real-world communication systems. In particular, we explore the use of *process calculi*, a well founded set of techniques specially designed to study the interaction and evolution of processes over time, to model and verify communication protocols for Peer-to-Peer (P2P) systems.

1.1 Motivation

Security of information has always been one of the main concerns in social behavior. The assurance of a personal secret which cannot be revealed to someone unauthorized, and the notion of trust have been relevant concerns since the beginnings of commerce and wars. The emergence of global communications, electronic processing, and distributed computation have increased the relevance of these concerns. Recent data from the 2004 Internet Fraud Crime report [CoI05] is just one example of the strong influence secure communications have in business: about 207.449 complaints (with quantitative losses of US$68,14 Millions) were reported to be related with threats including electronic fraud, identity theft and supplantation, and even hacking.

A wide variety of (automated) tools have been developed to overcome security risks, including firewalls, access control mechanisms and cryptographic-based software. These mechanisms by themselves, however, are not enough to provide security warranties; the open nature of the communications, and the inherent vulnerabilities of distributed systems makes it essential to provide higher levels of assurance for principals involved in a privacy-sensitive communication process. As a response to this problem, a set of methods known as *security protocols* have arisen: they define a precise set of steps that principals have to follow in order to establish secure communication between parties involved.

Security protocols have been widely used since its appearance, being at the heart of a huge amount of computer applications. However, we can never be confident over the security of a system unless we have some assurance of their effectiveness. As an example, one of the classical methods dates from 1978 when Roger Nedham and Michael Schröeder designed a protocol to prevent the disclosure of identities in an authentication process over untrusted networks such as the internet [NS78]. The protocol, apparently correct, was rapidly adopted in industrial and military applications until Gavin Lowe showed a flaw where messages in transit can be discovered and manipulated using a well defined set of steps [Low95]. With these results, one can question: How to ensure the correctness of a protocol?

Formal methods constitute an analytical approach for software and hardware design, that intends the reduction of errors by relying on solid mathematical models. One of the major benefits of formal methods is that they offer reasoning techniques that cover every possible state of a design, and the inclusion of well-defined proof techniques which ensure the accuracy and correctness of a design. The generality of formal methods contrasts with the ad-hoc spirit present in other approaches, such as empirical analysis and simulations. *Process calculi* constitute a particular class of formal languages, specially oriented to the analysis of concurrent systems. The main idea underlying process calculi is the abstraction of real systems in terms of basic units known as *processes*. The calculi provide precise elements to describe systems as a combination of processes, as well as offer tools to study the behavior of systems over time.

Consequently, process calculi appear as convenient tools to give a formal flavor to complex, concurrent computing systems. Several process calculi have been proposed over the last twenty years [Plo81, Mil95, Mil99, CG98, Hoa83, RP91]: although they differ on particular aspects for understanding communications, all of them agree on the basic principles given above. Following an interesting evolution, in the last five years process calculi have *particularized* in specific domain areas. In this way, for instance, several process calculi tailored for modeling biological phenomena have been proposed [RSS01, KD03, RPS$^+$04, Car04, BC02, GPR05]. Similarly, security has been a particular active area in this recent evolution: diverse process calculi, offering alternatives to the problem of modeling and verifying secure communications, have been proposed. Instances of these calculi include π and the Spi calculus [Mil99, AG99], the CSP process algebra [Hoa83], and more recently, the secure protocol language (SPL) [CW01].

This thesis explores the use of a process calculi in the analysis and verification of security protocols, providing an analysis of recently proposed models and tools, as well as contrasting their applicability in the modeling and verification of real world communicating systems. In particular, we focus on the study of communication protocols in Peer-to-Peer (P2P) systems. These systems, usually operating over open and distributed networks, take advantage of vast communication networks to accomplish diverse tasks in a very flexible manner. Examples of P2P communication systems include instant messaging applications, resource sharing web communities and collaborative work environments, such as MSN messenger [Ese02], Skype [BS04], Kazaa [GK03], Minerva [BMWZ05] or Gnutella [Rip01].

As in other contexts, the current ubiquity of P2P communication systems makes them prone to serious security vulnerabilities. Mainly because of their novelty, little work has been exercised in order to give formal warranties of security propierties in P2P systems. Our work intends to give concrete contributions in this context by studying two P2P communication protocols using SPL. MUTE, the first protocol, constitutes a flexible scheme for resource sharing in distributed environment. Security vulnerabilities for MUTE are identified and corrected. In the same sense, FTN, a collaborative P2P communication protocol is formalized and analyzed. Specific features in FTN lead to the design of encodings that ease the formalization of certain aspects present in other process calculi, important for the correct modeling of several concurrency protocols.

1.2 Objectives

General Objectives To explore the expressiveness of a security process calculus by means of modeling previously non-formalized real life communication protocols.

Specific Objectives

- To explore and analyze the nowadays existent process calculi concerned to security matters.
- To identify the most relevant features a process calculus must fulfill in order to model and verify systems related to security.
- To identify, select and justify the most appropriate process calculus for modeling and verifying secure systems.
- To understand about the different secure communication protocols, their basic phases and their implementation methods.
- To identify, select and justify two peer-to-peer protocols used in real life implementations, taking in count their functionality and the security properties they should fulfill.
- To verify the fulfillment of some security properties in three selected communication protocols, under the chosen security process calculus.
- To study an extension to the chosen calculus, in order to increase its expressive behind communication protocols.

1.3 Contributions

The main contributions associated with this work are presented below:

1. We give a comparative analysis of the most relevant process calculi concerned to security. Factors that influenced this comparison included syntactic structure, associated operational semantics and proof techniques.
2. We bear witness of the applicability of the SPL process calculus and its inherent proof techniques for modeling and reasoning about real life protocols.

3. By means of an SPL specification, we present a first formal characterization of P2P systems. Flexibility of the calculus allowed the inclusion of an specific set of roles, and considerations about the security issues related to every entity involved.

4. We provide a set of proofs related to two security protocols used in P2P systems, including the assurance of security properties never formalized in SPL. These proofs were formally derived from the process calculus specifications.

5. We propose and verify improved versions of the analyzed communication protocols, that correct security flaws (identified with the help of formal specifications).

6. We propose a set of encodings for SPL, that ease the description of certain kinds of protocols. These encodings are conservative with respect to the language.

It is worth pointing out that part of this work was presented as a contribution for *The Association for Logic Programming Newsletter Digest* [ALR05], reflecting some of the results associated with the MUTE protocol obtained in chapter 3.

1.4 Document Structure

The document is structured as follows: In the next chapter we present a brief description about the fundamental notions of communication and the importance of several approaches developed for describing and analyzing concurrent communication systems, such as process calculi. We make a particular emphasis in security concerns bearing witness of its relevance for communication environments, as well as giving the basis for those process calculi specialized with security, such as CSP, Spi and SPL.

In chapter 3 we show how SPL is a well suited framework for analyzing security aspects in P2P protocols, by modeling and verifying MUTE, a popular file sharing P2P protocol. We first analyze this protocol in order to ensure secrecy in an environment with outsider attackers which cannot get inside the network, and then we include new contributions to the P2P system in order guarantee a much more stronger property, such as secrecy behind an intruder which can masquerade as a trusted user.

Chapter 4 gives two different approaches for formal description and verification of P2P collaborative protocols. It presents two ways of modeling these kind of protocols. The first one includes a set of encodings representing new constructions for SPL syntax and the second one regards the development of a new protocol which extracts concepts from other different protocols. In our last case, we verify security properties such as secrecy and integrity.

In the last chapter we discuss some related work and we give out some concluding remarks, as well as pointing out to principal directions derived from this work.

2 Security in Communications

This chapter aims to introduce the reader into the main concepts of communication in computing, from basic models of sequential interaction up to more complex systems where concurrency and parallel computation play important roles. Here, several concepts are given in order to analyze properties concerned to the behavior of systems in which concurrent entities are in constant interaction. In the same way, this chapter has the objective of presenting the importance of security issues in communication systems, as well as giving a notion of some formal mechanisms developed for modeling and analyzing security properties in this kind of systems.

2.1 Communication

Communication is a wide concept covering several aspects of real life. From a baby crying at his mother's ear, to the interaction between a person with a computer machine, and much more complex interactions such as the ones involved in vast networks where data constantly flows. In this sense, communication could be defined as the process by which two entities exchange information through a medium, via a common system of symbols, such as sounds, words or even numbers.

In the following example, we have two entities which communicate between them.

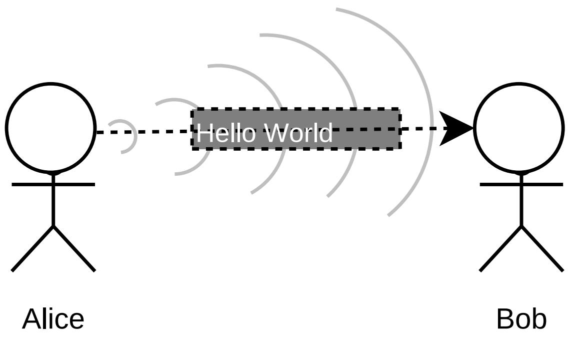


Figure 2.1: Basic Communication Process

Alice sends a message to Bob through the medium. In this particular case the message flows through a public channel until Bob receives it. The public channel resembles a channel with no access restrictions for any entity. Hence, we denote the medium as the public channel, since anyone can have access to it.

Although sending and receiving a single message can be denoted as communication, there are much more complex and specific ways of establishing a communication channel between principals in our daily life, known as *communication protocols.* These kind of protocols are a convention or standard, which control and enable the connection, communication and data transmission between entities.

In the following example, we present simple communication protocol in which two entities communicate between them in an interactive way, in order to answer a particular question requested by one of the participants. We can see series of steps both entities have to follow for succeeding in their intention to achieve the desired aim. This example, despite being a very simple communication protocol, is very useful to understand the meaning of the interaction by the movement of events.

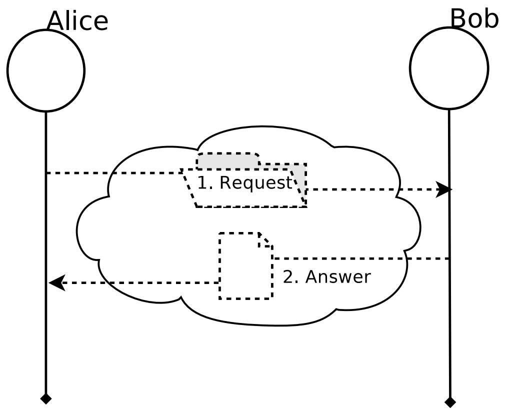


Figure 2.2: A simple communication protocol

Here, as an initial step, Alice sends a request message to Bob through the medium. By means of this message Alice asks a question to Bob. Bob receives the message and understands what Alice is asking for, and as a second and last step, Bob responds with a message answer to Alice's request.

Communication protocols are very common under several circumstances: from a pretty simple cellular interaction process taking place in our own body, to a single conversation between a pair of principals, where they have to follow series of steps in order to establish a comprehensive dialog, up to complex banking transactions.

The way in which two entities exchange information in an empty space, where there are no sounds or interruptions which may alter the communication sense, seems really intuitive and straightforward. However, is rarely feasible, there are several circumstances where communication cannot take place between two isolated objects; occasions where a set of individuals try to communicate at the same time, and some kind of order and control is needed, in such a way that all the participants can understand the information intended for them. That is when a previous important requisite must be fulfilled, which is the synchronization of events, in order for the communication to take place. This kind of action, where events occur at the same time and a previous synchronization phase is required before communication, is denoted as *concurrency.*

2.1.1 Communication in Computation

As stated before, there is a huge amount of fields where communication is present. Computation is one of those areas where communication plays a very important role, since, even in isolated machines, processes must establish series of interactions between one and another, in such a way that they may be able to work adequately.

In addition we can say that, since nowadays computing systems involve a lot of interaction between their components which are concurrently active, there arises the need of an underlying model with some basic inherent concepts, by which interactive behavior may be described and analyzed in such complex scenarios.

Communication systems such as computer networks, are on the need of understanding its behavior in a way that many different properties may be verified with respect to some form of ideal characteristics, so correctness in the designs or models beneath the systems can guarantee its well functioning. That is when the concept of formal models for concurrent communications arose, playing a very important role in computing, allowing the development of comprehensible models focused in specific phenomena.

2.1.2 Formal models for concurrent communication

Approaches for studying and building concurrent communication systems are not a novelty in computer science. There have been previous studies based on observations and ad hoc analysis, which have given an overview about the general concepts in which this kind of systems were based, their behavior, and the properties they should fulfill in order to guarantee their well functioning. These initial approaches have brought to life relevant theories, such as the ones related to semaphores, monitors or threads [SPG91].

Even though these were interesting works indeed, they did not enable efficient and understandable verifying mechanisms. A major problem, since verification of properties in these initial approaches seemed really complex and certainly very prone to errors. That was the moment in which the need of a formal model arose, in order to represent the interaction occurring inside these systems, enabling a more intuitive description of their behavior, and in consequence, more formal verification principles. These kind of models such as strand spaces, modal logics or process calculi appeared by means of several basic concepts extracted from other approaches such as graphs, Petri nets, transition systems and mathematical and logical theories.

2.1.3 Process Calculi

Process calculi cover the family of related approaches to formally model concurrent systems. The main idea underlying process calculi is the abstraction of real systems in terms of basic units known as *processes*. The calculi provide precise elements to describe systems as combination of processes, as well as offer tools to study the behavior of systems over time, providing a high level description of interactions, communication, and synchronization within them. Process calculi also provide a set of laws that allow process descriptions to be manipulated and analyzed, permitting a formal reasoning about equivalences between processes, such as those required for solving the classical problem of determining if an implementation of a protocol satisfies an ideal specification.

A process calculus has several important features by which it stands out over other formal approaches. It must have a syntax by which its constructs can fit an intended phenomena such as determinism, parallelism or recursion. It shall include a well established semantics which can give a meaning to the possible constructs inside the calculus. It requires mechanisms for comparing processes, as well as a way to specify and prove properties concerned to behavior of processes in a system.

We will present these ideas, by describing CCS, a simple process calculus defined by Robin Milner at [Mil95]. This calculus will enable us to analyze both sequential and concurrent processes in synchronous communication systems. We will present a brief overview of the way in which this particular calculus cover the issues a process calculus must fulfill.

The expressions of this language are interpreted by means of a labeled state transition system, which denotes a set of states, not necessarily finite, connected by labeled transition relations between its components. The transition system is the base of this particular calculus, since its transitions can capture the way in which new processes can be derived from others, via a particular action (Each state represents a process and a labeled transition, an action).

2.1.3.1 CCS Syntax

There are some important sets which will serve as the base of the entire calculus.

Let $N = a, b, ...$ be the set of names denoted as the output communication channels and $\bar{N} = \bar{a}, \bar{b}, ...$ its complementary set denoted as the co-names. If $\bar{a} \in \bar{N}$ and $a \in N$ where a and $\bar{a}$ are complementary actions, an interaction between parts of a process can be shown; The set of labels $L = N \times \bar{N}$ ranged over $(l, l', ...)$; And a set of actions $Act\ L \cup \{\tau\}$ where we call τ as an unobservable or silent action. These actions range over $(\alpha, \beta, ...)$

The syntactic set is defined as follows:

$$P, Q, ... := 0 \,|\, \alpha.P \,|\, P \parallel Q \,|\, P\ +\ Q \,|\, (\nu a)P \,|\, A\langle a_1, ...a_n\rangle \tag{2.1}$$

where capital letters $P, Q, ...$ act as process identifiers; 0 represents the particular nil process which does nothing; $a.P$ is a prefix, where P cannot proceed until action a is achieved; $P\|Q$ is the composition of processes P and Q, where they can proceed concurrently; $P + Q$ is a summation, where only one of the process involved can evolve; $(\nu a)P$ means that the channel a is restricted to the scope of process P; and $A(a_1, ...a_n)$ is the process A with parameters $a_1, ..., a_n$

2.1.3.2 Concurrent Processes

Every channel a is complemented by a channel $\bar{a}$. This is an important concept for communication, since they will represent communication channels. Every complementary pair $(a, \bar{a})$ will denote a possible interaction. In this way, each one of those pairs represent a synchronized action or a handshake, by which two processes will establish a communication channel. So we will say that a process transition is denoted by $P \xrightarrow{\alpha} P'$ where $\alpha \in \{a, \bar{a}\}$, α is the capability of process P to participate in a reaction with another process running concurrently and which can perform the complementary action. We say that both actions $a, \bar{a}$ are observable actions, and an interaction at a, is a mutual observation. In the same way, there is an special case where $\alpha = \tau$ will correspond to an unobservable reaction.

After defining the operational semantics in the following section, we will give out an example, recalling Alice and Bob, which will clear out the concept of reactions between processes in the CCS calculus.

2.1.3.3 Operational Semantics

$\frac{}{\alpha.P \xrightarrow{\alpha} P}$(ACT)	$\frac{P \xrightarrow{\lambda} P' \; Q \xrightarrow{\bar{\lambda}} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}$ (REACT)	$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$ (SUM$_1$)	$\frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$ (SUM$_2$)	$\frac{P \xrightarrow{\alpha} P'}{P\|Q \xrightarrow{\alpha} P'\|P}$ (COM$_1$)
$\frac{Q \xrightarrow{\alpha} Q'}{P\|Q \xrightarrow{\alpha} P\|Q'}$ (COM$_2$)		$\frac{P \xrightarrow{l} P' \; Q \xrightarrow{\bar{l}} Q'}{P\|Q \xrightarrow{\tau} P'\|P'}$ (COM$_3$)		$\frac{P \xrightarrow{\alpha} P'}{new\ a\ P \xrightarrow{\alpha} new\ a\ P'}$ if $\alpha \neq a$ and $\alpha \neq \bar{a}$ (RES)
$\frac{P_A[b_1,...,b_n/a_1,...,a_n] \xrightarrow{a} P'}{A\langle b_1,...,b_n\rangle \xrightarrow{a} P'}$ if $A(a_1, ..., a_n) \overset{\text{def}}{=} P_A$ (REC)				

Table 2.1: CCS Operational Semantics

Example: Let us define

$$Alice \overset{\text{def}}{=} \bar{b}.Alice' \text{ and } Bob \overset{\text{def}}{=} b.Bob'.$$

We can see that both have complementary actions, and if they are composed in parallel, they will react. In this way we represent the communication between two agents which have to interact so they both can continue working. So having $Alice \,\|\, Bob$, we will apply the transition rules described in 2.1.3.3 to describe their reaction:

1. Having $Alice \overset{\text{def}}{=} \bar{b}.Alice'$ and $Bob \overset{\text{def}}{=} b.Bob'$ we use ACT to infer $Alice \xrightarrow{\bar{b}} Alice'$ and $Bob \xrightarrow{b} Bob'$.

2. Then, using $REACT$ with $Alice \xrightarrow{\bar{b}} Alice'$ and $Bob \xrightarrow{b} Bob'$ we get the reaction $Alice \| Bob \xrightarrow{\tau} Alice' \| Bob'$

At the end of the transition we can see that a reaction has occurred and both processes, *Alice* and *Bob*, can evolve to *Alice'* and *Bob'*

2.1.3.4 Reasoning Techniques

Reasoning techniques are one of the most important characteristics in a process calculi, since with them a relation between processes can be established, enabling different ways to compare processes, as well as other manners to relate a real process with its abstract specification.

In CCS there exists the notion of equivalences which enable the reasoning about the behavior of processes. Here we present some of these concepts.

Definition 1 (Strong Simulation)**.** Let T be a transition system. A relation $R \subseteq S(T)$ is a simulation iff for every pair of states $(p, q) \in R$ and $p \xrightarrow{a} p'$ then exists some q' st. $q \xrightarrow{a} q'$ and $(p', q') \in R$

We can say that a state p **simulates** q if there exists a simulation R such that $(p, q) \in R$.

Definition 2 (Strong Bisimulation)**.** Finally, a *Strong bisimulation* (denoted by $\sim$) is considered as a relation where R and its converse R^{-1} are both simulations

Weak Bisimilarity In principle, two processes should be equivalent if no other process in the environment can see any difference in their behavior. This kind of bisimilarity captures this notion, since it relies in the equivalence between processes with equivalent observable actions.

For example, $\tau.P$ and P are not strongly bisimilar, but as we have said, they are weakly bisimilar since the equivalence is just focused in real actions from the observer point of view.

Definition 3 (Weak Simulation)**.** S is said to be a weak simulation iff for every $(p, q) \in S$ if $p \overset{e}{\Rightarrow} p'$ then there exists q' st $q \overset{e}{\Rightarrow} q'$ and $(p', q') \in S$ Where $\overset{e}{\Rightarrow}$ is as an experiment and denotes a sequence of observable actions interspread with zero or more unobservable τ actions.

Definition 4 (Weak Bisimulation)**.** A weak bisimulation ($\approx$) is a binary relation S over processes where both S and its converse are weak simulations. We say $P \approx Q$ when P, Q are weak bisimilar. Intuitively we can say that every strong bisimulation is a weak bisimulation.

2.2 Security

Several aspects of life involves security, basically the ones that involve personal information that can be manipulated or missused. It is very common to hear about banks robberies, identities supplantation or even frauds. Many of these threats were available due to obvious infrastructural problems, like the absence of reliable communication methodologies, alert systems, and naive behavior between involved parts.

As time evolves, techniques to overcome these problems have arisen: big robberies are less frequent due to the creation of complex alarms, communication systems and transportation methods which improve the effectiveness and time of response for security corps, protecting large organizations; a wide variety of reliable identification systems have been created in order to verify individual identities, and trust certificates have become an useful policy in order to support confidence in commercial activities. Unfortunately, intelligence of attackers have become more sophisticated, as well as their techniques and tools. The emergence of computer systems have been a cornerstone in the development of new attacks, from simply brute-force analysis of passwords, to highly technical attacks involving distributed agents that contribute to each other in the search of a security hole of a system, passing through computer viruses and malware [1].

In this way, searching for solid foundations in security has become a major problem in nowadays. Detaching each aspect involved in this topic, there are three main levels that a system might consider in order to achieve security in environments: The first one, concerned with the basics primitives of a system, is devoted to algorithms and intends to guarantee that the tools of a system themselves are capable enough to resist security attacks from an intruder that knows characteristics of a system; this level includes important security areas like cryptography, random number generation, secure channel creation and so forth. The second one appears when distributed systems are involved, because communications between agents open new opportunities for the attackers, even having the protection of cryptographic algorithms in the systems. Finally, the third level specifies security goals of the system, and define which protocols and tools are useful to tackle these goals [ASL00]. In this section, we will introduce the main characteristics of each of these topics, in order to show the relevance for communication systems. In this order of ideas, we present the actual main concerns in security, as well as going in detail with each one of the security levels, showing their main features as well as the classical techniques.

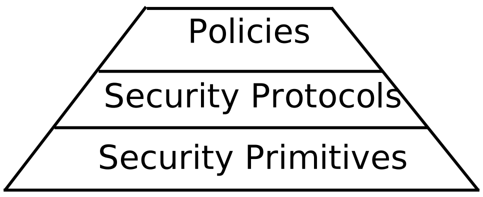


Figure 2.3: Levels of Security in a System [ASL00]

[1]Hardware, software, or firmware that is intentionally included or inserted in a system for a harmful purpose. Available at http://www.ee.oulu.fi/research/ouspg/sage/glossary/

2.2.1 Security Properties in Communication Systems

Establishing a rigorous direction of needs in nowadays communication systems is not an easy task. Assertions about the ”correctness” or ”security” of a protocol are vanished, primarily because of the vast variety of purposes by which these systems are directed. As an example, a communication protocol operating in a Peer to Peer (P2P) system intends to guarantee that the information over the network is preserved despite the permanent movement and lack of persistence in the connection of the principals involved; however, this need is addressed in a completely opposite way comparing it to a transactional system which only accepts a message if all agents involved are present along the entire protocol run [BP01]. In this way, some design considerations in the field of security have to be presented prior to the selection of a secure communication system [Aba00]:

- *Open Networks:* Due to the vast connectivity of systems, more resources and users are available in the same environment, so overload and corruption become very common problems. Even worse, due to the ability of users to access more systems, access controls for single users may be inadequate in open networks.
- *Dynamic Configuration:* In a communication process between agents in an open network, several paths are available to route each of the messages involved, therefore the security of the system may need to ensure that using every path to route a message, maintains private messages safe.
- The principals involved in a communication network operate over a wide and open environment, in presence of not only trustful neighbors, but hostile agents that can behave in the opposite direction of the principal purpose of the protocol, modifying the information involved in each interaction. Moreover, it is possible that these agents are not completely trustful, a dangerous risk if the system shares critical information.
- The analysis of security properties is almost completely based on the resources available to attackers. Sometimes is necessary to model fraudulent agents in uncommon scenarios, these scenarios represent almost the worst case that a protocol can be exposed. As an example for a well known protocol like the Needham-Schroeder [Low95], it is necessary to suppose an attacker present in every interaction between agents.
- Security Assurance might involve abstracting the communication model from other desirable properties in a computing system, like correctness and efficiency.

With these considerations, the assurance of communication protocols often used a number of *security properties* defined. The meanings of these terms are frequently taken as obvious and widely understood, but it often turns out that for these notions, different kinds of interpretations are given, even in a single document. For this reason is crucial to give a precise and formal meaning, since when specifying a protocol it can only be claimed as correct or secure when compared to a precisely defined property.

In the following lines we present a number of general properties used in the verification of communication protocols, stressing the fact that the selection of desirable properties for each protocol are related to its specific use.

- *Authentication:* It is declared for two specific purposes: to bring credibility to a message received [Sch96c], and to assign responsibilities in communication tasks [FG01]. In this way, authentication gives the system the ability to make decisions about the correct identification of an individual, or to verify that an agent involved in the communication protocol is the correct person he intends to be.

- *Confidentiality:* To ensure that resources in a system are protected from unauthorized access and use of principals strange to the system. This is a very common property in the design of servers and firewalls. However, sometimes the controls established by these devices are intentional broken, and other access controls are required[RSG+01]. Another important property derived from confidentiality is known as *Secrecy* [Aba00, MKL+02], which intends to guarantee that the privacy-sensitive information of a system will not be revealed to any unauthorized principal.

- *Availability:* This property ensures that the resources of a computer system are available to authorized agents [Gol99]. Availability tackles different kinds of threats, like overflow attacks (Denial of Service -DOS-), buffer overflows and electrical accidents.

- *Responsibility:* States that every action on the computing system can be traced up to the agents that originated it [Aba00]. One variation of this property, known as *Non-Repudiation [BP01]* states that an agent receiver of information has the ability to prove that the sender of some data did in fact send it even though the sender might later desire to deny his actions.

- *Non-Malleability:* This property establishes that the agents involved in a protocol cannot modify the contents of a given message. This property, fundamental in transaction systems can sometimes be took for granted in communication protocols that involves cryptography, supposing that without the appropriate key, an attacker cannot modify a message without corrupting its structure [Rud00]. A variation of this property is known as *Integrity.* There are two main definitions of integrity in computing. The first one consists in ensuring that a computing system is protected against the unauthorized manipulation/destruction of data [RSG+01]. The second definition addresses to computer system correctness, establishing that those systems must behave in the way that they are constructed for, avoiding malfunctions [Gol99].

- *Anonymity:* Establishes that the credentials of each agent in the system are only known by authorized agents, meanwhile unauthorized principals can read their published messages but cannot know their identities [MKL+02]. One derivation of Anonymity is *Non-traceability,* a property which implies that is impossible to determine the origin of a message in transit on the network. This property is desirable in P2P networks, and its purpose is addressed in the opposite that address authentication.

Is an obvious fact that communication systems by themselves can not fulfill these requirements. That is why a wide variety of tools are created to complement the accomplishment of these tasks, using cryptography as the cornerstone where security lies. In the next section, we shall explain some of these tools and relate them to security concepts.

2.2.2 Cryptography

Cryptography itself deals with the communicating presence of adversaries. In this way, previous messages passing in a medium would be available for every agent in the network, a non-grateful characteristic for task compromising sensitive information. Examples of these systems were present in World War II, where different armies of opposite sides shared a common medium (Radio channel) in order to transmit tactics that can reveal the next movements of troops. To tackle this problem, several techniques were developed in order to accomplish trusted communication between certified agents.

We shall introduce some basic concepts that will be used later in this chapter:

Definition 5 (Messages in a Cryptosystem)**.** Let P a set of *Plaintexts* (an ordinary message completely visible and understandable in natural language), C a set of *CipherTexts* (Completely non-understandable messages, where no information can be obtained/inferred from its contents) and K as a set of *Keys* (Parameters that, combined with a function, produces a ciphertexts from a plaintexts). Finally M is defined as a message where $M \in \{P \cup C \cup K\}$ $\square$

The first case where history reports the use of cryptographic techniques raises in the middle of the Roman empire. The emperor Caesar, worried about the correct execution of his orders, created a simple technique that prevented that the messages sent across the empire can be discovered by unauthorized people. The basic idea was to share a common numeric key k between the generals and the emperor, shifting the order of the letters k positions below, and transmitting the message over the imperial courier. Finally, when the message is received for the correct principal, he just has to switch backward the message k positions in order to discover it.

$$\begin{array}{ccc} m & \xrightarrow{C(m)} & n \\ ''\textit{Attack Constantinople}'' & \overset{3}{\to} & ''\textit{Dwwdfn Prqvdqvlqod}'' \end{array}$$

In this example, we use the Caesar's code with to convert a plaintext m to a ciphertext n with a function (a simple substitution) using the key k (with a switching factor of 3). It is clear at first sight that the message is unintelligible for every person without knowledge of the underlying technique for discovering the data; but with a little logic sense, a good observer can search for patterns that can break the message. This brief example introduce us to the first model of cryptographic techniques, known as *Symmetric Cryptography.*

2.2.2.1 Symmetric Key Cryptosystems

Basically, the systems that use a single key for encryption and decryption of messages are known as *Symmetric key Cryptosystems* [Hut01], as an example Alice and Bob must establish a previous agreement in order to sign every message in transit with a key in common, using practically the same algorithm to transform a message from cleartext to ciphertext and viceversa.

Definition 6. Symmetric Key Cryptography:

Let A, B agents, $key(a,b) = key(b,a)$ a shared key between principal A and B, m a message in plaintext and c a ciphertext. Symmetric key encryption/decryption functions are defined as:

$$\begin{aligned} c_i &\overset{def}{=} sc(m_i, key(a,b)) \\ m_i &\overset{def}{=} sc^{-1}(c_i, key(a,b)) \end{aligned}$$

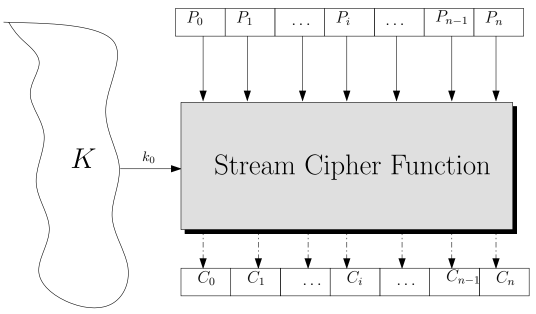
Where $sc(\dots)$ and $sc^{-1}(\dots)$ are essentially the same operations or algorithms. □

There are two main classes of symmetric key Cryptosystems, characterized by the way that they encrypt messages: *Stream Ciphers* [Rue86] and *Block Ciphers* [Lai92].

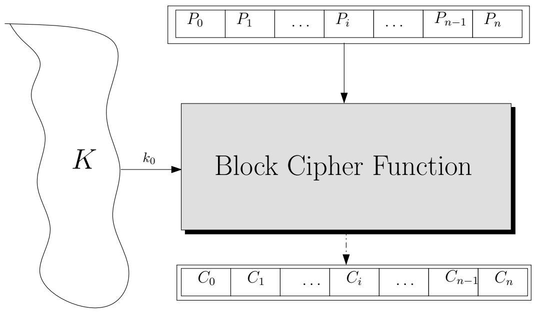
- *Stream Ciphers:* The basic characteristic of this class of ciphers lies in the way they encode the message. The main idea is to divide the contents of the message as a sequential composition of tiny messages replaced or substituted for ciphertexts of the same length generated by a substitution key (see image 2.4(a)). This approach has advantages in terms of the leverage of computational power used in order to encrypt or decrypt a message, but also has very important limitations to consider: the first one deals with the error control of messages, because the atomicity of each message can be corrupted if at least one of the submessages are lost or modified. The second one involves a crucial security risk, symmetric key cryptosystems are based on the assumption that unauthorized agents can never have a secure key known by every trusted user. This assumption is very difficult to prove in a practical environment, as well as addressing other topics outside cryptography itself like the correct management and distribution of keys.

- *Block Ciphers:* These cipher algorithms treat a message as a whole entity, splitting the contents of the message into blocks and permuting them using a transforming function that encrypts the whole block, converting into pieces of ciphertext that can only be understood with the entire message and the correct key, as image 2.4(b) shows. Sometimes, the simple substitution was not enough to obscure the relation between plaintext and ciphertext, so substitution boxes (*S-boxes* [Hut01]) came as a mechanism to prevent the disclosure of the messages. This mechanism acts as a look-up table were a sequence of bits are dynamically converted to a different one using a pattern. Block cipher algorithms has some advantages over stream ciphers, like control correction and integrity.

However, in many systems both approaches are mixed together in order to achieve a suitable symmetric key cryptosystem, using block ciphers in an authentication phase and later on using stream ciphers for transmission of messages.



(a) Stream Cipher Encryption



(b) Block Cipher Encryption

Figure 2.4: Types of Symmetric Encryption

Although symmetric key cryptography has well known advantages, such as the efficiency in time and computing power, its a difficult task to guarantee that agents involved in a communication protocol might be trustful enough to kept shared keys away from untrustful agents. In this way, symmetric cryptography is specially used in closed environments, where we can manage some environments like the number of agents involved or the communication channels.

2.2.2.2 Public Key Cryptosystems

Sometimes communication is about open environments increasing the risk for a leakage of security using shared keys. Imagine for example two people, Alice and Bob, sending a secret message through public mail service. In this example, Alice has the secret message and wants to send it to Bob, after this, Bob sends a secret reply (see image 2.5).

With a symmetric key cryptosystem, Alice and Bob arrange a previous meeting in order to create a common key for both of them, then Alice places the secret message on a shelter, and locks it using a padlock with her key. She then sends the box to Bob through regular mail. When Bob receives the box, he uses an identical copy of Alice's key to open the box, and reads the message. In this way, Bob can also use the same padlock to send his secret reply.

In a public/asymmetric key system, Bob and Alice have separate padlocks. First, Alice asks Bob to send his open padlock to her through regular mail, keeping his key hidden to public, only available to himself. When Alice receives the padlock, she uses it to lock a shelter containing her message, and sends the locked shelter to Bob. Bob can then unlock the box with his own key and read the message from Alice. To reply, Bob must similarly get Alice's open padlock to lock the box before sending it back to her.

More formally, a public key Cryptosystem [Hut01] is defined as follows:

Definition 7. Public Key Cryptography:

Let A, B agents, $Pub(a)$ a public key for A, $Priv(A)$ the secret key from A, m a message in plaintext and c a ciphertext. The public key encryption/decryption functions are defined as:

$$\begin{aligned} c &\overset{\mathbf{def}}{=} E_{Pub(A)}(m) \\ m_i &\overset{\mathbf{def}}{=} D_{Priv(A)}(c) \\ &\text{and} \\ m &= D_{Priv(A)}(E_{Pub(A)}(m)) \end{aligned}$$

Where $Priv(A))$ must be completely undeducible from $Pub(A)$. □

The critical advantage in an asymmetric key system is that Bob and Alice never need to send a copy of their keys to each other. This substantially reduces the chance that a third party (perhaps, in the example, a corrupt postal worker) will copy a key while it is in transit, allowing a third party to spy on all future messages sent between Alice and Bob. Another advantage is present in the ease of key distribution, allowing an agent to publish his own key in a public site without need of previous agreements. However, if Bob was careless and allowed someone else to copy his key, Alice's messages to Bob will be compromised, but Alice's messages to other people would remain secret, since the other people would be providing different padlocks for Alice to use.

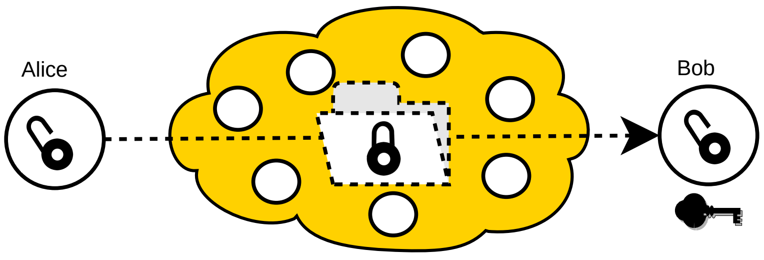


Figure 2.5: Public Key Cryptography

2.2.2.3 Digital Signatures

Ciphers are not only used for guaranteeing secrecy properties, but also used to guarantee authentication of each of the agents involved in the protocol. In this way, the agents involved will trust that the information received from an agent A is really generated by him, without unauthorized modifications (*fraud*) or threats from other agents (*phishing*[2]) . For example, consider an e-commerce application and you want to ensure that the orders for every customer in your system are really placed by trusted users, avoiding thefts that impersonate trusted users. In this way, you must provide mechanisms of authentication other than user/password that can be stolen. In this way, digital signatures emerge as an ideal tool to achieve this. The

[2] "Phishing" is a form of Internet fraud that aims to steal valuable information such as credit cards, social security numbers, user IDs and passwords.

customers are able to certify each order signing it with an own key for private use, and you only have to check if the sign matches with the one that you have stored in your database.

We can use Public Key Cryptography (PKC) in order to achieve these tasks. To do so, the cryptographic scheme is subtle modified in his primitives.

Definition 8. Digital Signature Schemes:

Using PKC, we can include an inverse function of E st.

$$m = D_{Pub(A)}(E_{Priv(A)}(m))$$

Where $Priv(A))$ must be completely undeducible from $Pub(A)$. □

Doing so, if Alice must authenticate a message, she just has to encrypt it with his private key, and Bob only has to decrypt the document with Alice Public key in order to check that the message was correctly received (see figure 2.6). This increases the security of the system, relying on the assumption that Alice never publishes her private key to anyone else.

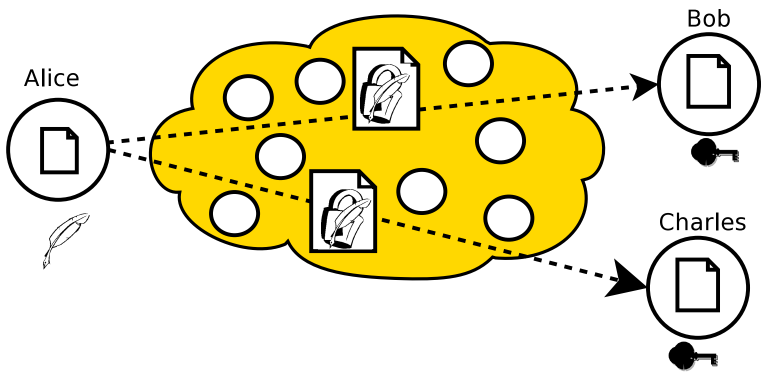


Figure 2.6: Digital Signatures

2.2.2.4 Hash Functions

Another widely used technique used to deal with integrity issues is known as *hash functions*. This approach uses a computational concept known as **one-way functions**. These set of functions are the ones that, given a function $f(x)$, it is practically infeasible to find a function $f^{-1}(x)$ st. it can be computable in polynomial time.

A hash function basically consists of a transformation of a message m to a message h, where h is a message of fixed length. Basically a hash function $H(m)$ has the following characterization:

- The length of m can be variable.
- $H(m)$ can be computed in polynomial time.

- $H(m)$ is a one-way function.
- $H(m)$ is collision-free. This means that, given $H(m)$, it is computationally infeasible to:
  - Construct a fixed message m' st. $H(m) = H(m')$.
  - Find an arbitrary message m' st. $H(m) = H(m')$.

A typical protocol that uses hash functions can be seen as follows: Alice wants to send a message m to Bob. She then sends two messages with $H(m)$ and $E_{Pub(Bob)}(m)$ respectively. In this way, Bob only has to decrypt the message received and apply the hash function to m and check its correctness with respect to $H(m)$.

As well as cryptographic tools are important to construct a system well suited for dealing with security issues, they can not ensure the security of the system by themselves. A common practice is to over trust underlying key exchange systems, or to forget the safe storage of the secret keys, open to the disclosure of a system. In this way, Security(or cryptographic) protocols emerge as a way to ensure the correct execution of a system. Concretely, a security protocol is an abstract protocol that performs a security-related function. It includes primitives for concurrent communication, as well as cryptographic operations in abstract way so they can be checked without concern of implementation details outside of the scope of these techniques. In this work, we aim to review some of the formalism that deals with the analysis of security protocols, from abstract models only used to denote threats, to well-founded logics suited to deal with security issues.

2.2.3 Dolev-Yao Model

One of the first formal approaches for modeling and analyzing security protocols was the model presented by Danny Dolev and Andrew C.Yao [DY81]. It is a simple and useful framework in which security protocols could be specified and verified in a very simple manner. Protocols in this approach are modeled in a clear notation, where $X \rightarrow Y : M$ means a message M sent from agent X to agent Y, and M could represent a plaintext m or a cyphertext $\{m\}_k$.

The Needham-Schroeder-Lowe protocol (NSL) is an authentication protocol, which describes the interaction between two agents (*Alice* and *Bob*). *Alice*, acting as the initiator and Bob as the responder. *Alice* sends *Bob* a fresh generated value among with its particular name *Alice*, both encrypted with *Bob's* public key. When Bob receives the message, he decrypts it with his own private key. Then Bob sends a message containing a new fresh name among with the name received from *Alice* and his own name, encrypted with *Alice's* public key. *Alice* recovers its fresh name and convinces herself that she has communicated with *Bob*. If this is true she sends to Bob the fresh name received from him encrypted with his public key. Then if *Bob* recognizes his fresh name, he can be sure that he has communicated with *Alice*.

$$\begin{array}{ll}
(1) & Alice \rightarrow Bob : \{m, Alice\}_{Pub(Bob)} \\
(2) & Bob \rightarrow Alice : \{m, n, Bob\}_{Pub(Alice)} \\
(3) & Alice \rightarrow Bob : \{n\}_{Pub(Bob)}
\end{array}$$

One of the most important values stated on this model are the assumptions presented, which certainly simplify the reasoning about security protocols. Here we present the general ideas about these assumptions exposed by Dolev and Yao in their work:

- Cryptography is unbreakable: This means that although a saboteur can eavesdrop a message, if the message is encrypted and the spy does not have the right decryption key, it would not understand the meaning of the message.
- Uniform Protocol: It means that the same protocol is used for each pair of agents which want to communicate
- Active Intruder: Intruders are active agents who can eavesdrop messages, masquerade as a trusted users and participate in the protocol sending and receiving messages.
- The intruder does not know the behavior of the protocol.

We recall these concepts, since they are of the essence of almost all the security process calculi we will present and analyze in our following section.

2.3 Process Calculi for Security Protocols

Security process calculi are those focused in modeling and verifying security issues in communication protocols. The π [Mil99] and the Spi calculus [AG97a], CSP [Hoa83] and SPL [Cra03] are one of those calculi which allow modeling this kind of properties related to security. This by means of the essential properties a secure process calculus must fulfill, such as cryptographic primitives and fresh names generation, among with the usual characteristics any common process calculus must have. Although CSP is not precisely a process algebra concerned to security, we can use it because several works demonstrate that, by means of CSP models based on abstract data types which represent cryptographic and fresh names notions several security protocols have been specified and verified successfully.

Probably the π and Spi calculus among CSP are ones of the most studied secure process calculi in the present time. That is why we will focus in a description of both of them, among with the SPL calculus, a recently proposed security protocol language based in the concept of Petri nets. We will use these four examples to give a general overview about security process calculi.

2.3.1 π calculus: Proving Security using secure channels

Milner proposed the π calculus as a well founded mathematical model that represents processes and their interactions over a dynamic environment [Mil99]. The basic idea underlying this calculus is the mobility of information; in this way, the processes can interchange information at the level of channels, allowing processes to access new resources over time. Such mobility inherits the security risk of communicating systems. The first attempt to formally verify security properties was done by Milner, Parrow and Walker at [MPW89]. They strongly used the notion of private channels in order to show that given a protocol, the channels generated by the participants involved were never eavesdropped by an outsider agent. In the next lines we are going to outline the basic concepts of π calculus, applying them to prove security properties in communication.

2.3.1.1 π Outline

Let $x = m, n, \ldots, x, y, z, \ldots$ be an infinite set of names, also known as communication channels, and $P = P, Q, R, \ldots$ a set of processes of the following form:

$$P ::= \bar{x}(y).P \quad | \quad x\langle y\rangle.P \quad | \quad (P|Q) \quad | \quad (P{+}Q) \quad | \quad (\nu x)P \quad | \quad !P \tag{2.2}$$

Where $\bar{x}(y).P$ and $x\langle y\rangle.P$ denotes the output and input process of a channel y over x, respectively. $P|Q$ denotes the concurrent execution of processes P and Q, $P + Q$ the non-deterministic choice over P and Q, $!P$ the endless execution of the process P. Finally one of the main elements in the π calculus to express security is the process $(\nu x)P$, which represents the restriction (or binding) of the variable x with a fresh, unique and randomly generated value known as *nonce* in the process P, and can be seen as the creation of a channel in the context of P.

π is devoted to processes and their interactions, so the semantics provides a clear definition of how the processes interact with each other and how terms are propagated. The operational semantics of π calculus is based on structural congruence and reduction rules, giving enough power to compare processes and show properties between them. With structural congruence, two processes can be compared statically showing syntactic or structural similarities between them. Structural congruence is useful to compare processes like $a(x).\bar{b}(x)$ and $a(y).\bar{b}(y)$, only different in the selection of the name, but completely equivalent in their behavior. With reduction rules we can trace how the interactions of each agent in the protocol affect the local knowledge of a process: in this way a process $(\bar{a}(x).P \quad | \quad a(y).Q)$ can be reduced in a subsequent event as $P \quad | \quad Q[y/x]$ where $Q[y/x]$ denotes the *alpha conversion* of the channel y with x.

Scope Extrusion The π calculus allows the mobility of channels, based on its interaction rules. In this way, the input and output processes $(\bar{x}(y).P \mid x\langle z\rangle.Q)$ describe how processes interact sending information over a public channel x. However, there are scenarios where the creation of new channels is needed to ensure fresh and private communications between agents, therefore $((\nu x)(\bar{a}(x).P) \quad | \quad a\langle y\rangle.Q)$ allows the channel x to broad his scope only for P and to be reached by Q using reduction rules. However, interaction itself does not guarantee secrecy properties, See the example below.

Example 2.1. Secrecy in the π-Calculus

Let Alice, Bob and Steve be agents, such that

$$\begin{array}{lll} Alice & \triangleq & \bar{x}(y) \\ Bob & \triangleq & x\langle z\rangle \\ Steve & \triangleq & x\langle z\rangle \\ and & & \\ P & \triangleq & Alice \mid Bob \\ P' & \triangleq & Alice \mid Bob \mid Steve \end{array}$$

In this scenario Alice sends to Bob a name y via a public channel x. It is insecure since anybody can receive message y through channel x. However, Steve can receive any message Alice sends to Bob. To avoid this situation, we restrict the channel x just to Alice and Bob in the following way:

$$(\nu x)(Alice \mid Bob) \mid Steve.$$

In this case the channel between Bob and Alice is restricted to them and Steve can not eavesdrop any message through it.

Using scope extrusion it is possible to model unguessable secrets in the π calculus, so the process that cannot access the channel will not known the secrets involved.

As we can deduce by the example, this notion of secrecy, can be assumed as perfect. Nevertheless, that is actually an inconvenient because the security of the model relies in how this channel may be modeled, with possible security breaches it may have and how these problems may be suppressed. Therefore, we need a less abstract concept, a model by which we could go closer to the implementation of security in communications, so we can understand the actual security protocol running underneath the private channel and the possible security failures it may present. That is when we can see the real importance of process calculi particularly focused in security.

From a practical point of view, implementing a secure communication channel between two points is not feasible, since there are no channels which can provide information transference without risk of interference or tampering. Although the concept of restricted channels is certainly an abstraction, it is an essential tool for bringing to real life something close to the concept of these channels.

2.3.2 Spi Calculus

The Spi calculus [AG97a], is an extension of the π calculus [MPW89] specially designed to deal with cryptographic protocols. As presented before, the π calculus is a fairly convenient formalism to describe concurrent communication, allowing to model security issues like authentication and secrecy in an abstract level. However, the π calculus does not include means to appropriately represent some security primitives commonly used in describing security protocols, such as encryption and decryption.

With this motivation in mind arises the Spi calculus, extending the π calculus with primitives for encryption and decryption, with a precise semantics that allows to reason about privacy or authentication in the protocols. More specifically, the security proofs in the Spi calculus are based in a set of equivalences and reduction rules.

2.3.2.1 Spi Syntax

The extension of the syntax in the Spi calculus is basically composed a set of terms, that can be names or variables, and a set of processes. The set of terms is defined by the grammar below:

$L, M.N, \ldots$		::=	terms
	$l, m, n, \ldots$		names
	$x, y, z, \ldots$		variables
	(m, n)		Pair
	0		Zero
	$suc(m)$		successor of m
	$H(m)$		Hashing
	$\{m\}_n$		Shared key encryption
	m^+		public key
	m^-		private key
	$\{[m]\}$		Public key encryption
	$[\{m\}]$		Private Key Signature

Table 2.2: Spi Terms

As messages can be composed by any number of components (polyadicity), the constructions of pairing must be included in the calculus without deeply extensions of the π calculus (see [MPW89]). The same argument is suited for the inclusion of primitives for integer treatment. The basic features of the Spi calculus are the inclusion of primitives for encryption, the handling of shared keys as standard names, and the use of public and private keys of a message m as m^+ and m^- respectively. The syntax provides the necessary constructions to express public and shared-key encryption, as well digital signing. The inclusion of hash function $H(m)$ without a reverse equation corresponds to the assumption that any message converted with a perfect hash function cannot be inverted.

The Spi calculus includes processes as another syntactic set in the grammar, which basically denotes the inverse behavior of encryption and decryption processes, as well as signatures verification. (See table 2.3)

$P, Q, R \dots$		$::=$	Processes
	$\dots$		As in equation 2.2
	$[m \quad is \quad n]P$		Match
	0		Nil
	$Let\,(x, y) = M\ in\ P$		Pair Splitting
	$case\ m\ of\ 0 : P\ suc(x) : Q$		Integer Case
	$case\ L\ of\ \{x\}_n\ in\ P$		Shared Key decryption
	$case\ L\ of\ \{[x]\}_n\ in\ P$		Public Key decryption
	$case\ L\ of\ [\{x\}]_n\ in\ P$		Signature Check

Table 2.3: Spi Processes

2.3.2.2 Spi Semantics

The π calculus, is based on a set of reduction rules which show how processes interact over time. However, the Spi calculus introduces a new set of equivalences and reductions that operate over processes with cryptographic primitives, representing how the *knowledge* of the system is modified over time. The foundation of these rules is the reaction relation introduced in [Mil99]; such a relation basically states how processes sharing a common communication channel in complementary processes can follow with their subsequent behavior. More specifically, given two processes acting in parallel, $m(M).P|\bar{m}(x).Q \longrightarrow P|Q[x/M]$.

This notion has been used to declare reductions in the Spi calculus, extending it to express synthesis and allowing to carry out reasoning about process evolution in a more convenient way. Being more concrete, we can see the reductions of the Spi calculus as the following rules for process of replication, matching, pair splitting, and decryption:

$!P$	$>$	$P\|!P$	Replication
$[M \text{ is } M]P$	$>$	P	Matching
$\text{let}(x, y) = (M, N)\ in\ P$	$>$	$P[M/x][N, y]$	Pair Splitting
$case\,0\,of\,0 : P\,suc(x) : Q$	$>$	P	Zero
$case\ suc(M)\ of\ 0 : P\ suc(x) : Q$	$>$	$Q[M/x]$	Successor
$case\{M\}_N\ of\ \{x\}_N\ in\ P$	$>$	$P[M/x]$	Decryption

Table 2.4: SPi reduction rules

Given these rules, a more formal notion of equivalence is stated, to show how a processes A and B, not always syntactically equivalent, can express the same behavior. The concept of *structural equivalences* can express these similarities using a set of rules, and the notion of reaction, which we can see in table 2.5.

$$\frac{}{P|0 \equiv P}\text{(Struct Nil)} \qquad \frac{}{P|Q \equiv Q|P}\text{ (Struct Commutativity)} \qquad \frac{}{P|(Q|R) \equiv (P|Q)|R}\text{(Struct Associativity)}$$

$$\frac{}{(\nu m)(\nu n)P \equiv (\nu n)(\nu n)P}\text{(Struct Switch)} \qquad \frac{}{(\nu m)0 \equiv 0}\text{(Struct Drop)} \qquad \frac{}{P \equiv P}\text{ (Struct Reflection)}$$

$$\frac{}{(\nu n)(P|Q) \equiv (\nu n)P|Q}\text{ if } n \notin fn(Q) \quad \text{(Struct Extrusion)} \qquad \frac{P > Q}{P \equiv Q}\text{ (Struct Reduction)}$$

$$\frac{P \equiv Q}{Q \equiv P}\text{ (Struct Symmetry)} \qquad \frac{P \equiv Q \qquad Q \equiv R}{P \equiv R}\text{ (Struct Transitivity)} \qquad \frac{P \equiv P'}{P|Q \equiv P'|Q}\text{ (Struct Parallel)}$$

$$\frac{P \equiv P'}{(\nu m)P \equiv (\nu m)P'}\text{ (Struct Res)} \qquad \frac{P \equiv P' \quad P' \to Q' \quad Q \equiv Q'}{P \to Q}\text{ (React Struct)} \qquad \frac{P \to P'}{P|Q \to P'|Q}\text{ (React Parallel)}$$

$$\frac{P \to P'}{(\nu m)P \to (\nu m)P'}\text{ (React Res)}$$

Table 2.5: Spi Calculus operational semantics: Structural and reaction rules

2.3.2.3 Security Proofs in the Spi Calculus

The Spi calculus provides two particular ways to cover security analysis: the first guarantees security properties relying on the concept of equivalences. In this way, properties like secrecy for a protocol P that keeps a secret information X are expressed stating that the instance of a protocol with the message X is equivalent to the protocol with X', for every run in the protocol and every message X'. The proofs consider an arbitrary environment where possible attackers can receive and forge information, including new messages in the network. This approach is strongly based on the elegant concept of structural equivalence, needing to relate every model to a sort of "magical", correct and secure implementation that does not disclose any message received, making the proofs rather complicated [AG97a]. To overcome these difficulties, a new set of semantic notions are introduced in the calculus. These concepts rely on the notions of bisimulations and an inductive characterization of reaction without appeal to structural equivalence.

2.3.3 CSP

CSP [Hoa83] is an abstract language for describing systems of concurrent agents which interact via message exchange. It is intended to be a multipurpose algebra: several specialized theories could be constructed on top of its semantic model. In this way, concrete formalisms can be designed and proved using this theory, with an environment especially crafted for each purpose. Security has not been a topic left away and several approaches for analyzing security properties in protocols under this framework have been developed. Later on we will show how this can be possible.

2.3.3.1 Syntax

Systems in CSP can be represented by processes which may interact with others via a series of events or actions.

Events: Actions or events are essential in CSP since they represent the interaction of processes inside a system. The set of all possible events in which a system may engage in is denoted as Σ. Events may be atomic in structure or may consist of several distinct elements. In this way events in CSP can consist of different types of components. This is an important issue, since several specialized theories such as security, requires working with abstract types such as encrypted or signed messages.

An intuitive example for describing events could be a simple vending machine which delivers sodas. Here we have two particular kinds of events: $Coin$: The insertion of a coin in the slot of the vending machine and $Soda$: The extraction of a soda from the dispenser. Then we say that the alphabet of this particular system $\Sigma = \{Coin, Soda\}$

Communicating events: These particular type of events are described by the pair $c.v$ where c denotes the name of a channel in which the communication takes place, and v is the value of the message which is intended to be passed through the channel. Particularly $c?v$ is defined as the input event in which the value v is received via channel c. While, the output event is represented as $c!v$.

Processes: These are the fundamental components of the calculus. The entities described using CSP by means of the events in which they may engage in.

These are the most common processes structures used in this calculus:

- *Stop* This is the process that cannot generates events at all. Represents a deadlock.
- $a \to P$ Being P a process, it is only able to initially perform a before continuing as P.
- $P \Box Q$ The process P choice Q can behave either as P or as Q.
- $\Box_{i \in I} P_i$ Indexed form of choice.
- $P \sqcap Q$ Non-deterministic choice.
- $\sqcap_{i \in I} P_i$ Indexed form of non-deterministic choice.
- $P|[D]|Q$ Parallel composition between P and Q processes with the requirement that they have to synchronize on any event that belongs to the synchronization set D.
- $P|[\{\}]|Q$ or $P|||Q$ Parallel composition with no requirements.
- $|||_{i \in I} P_i$ Indexed form parallel composition with no requirements

Processes in CSP can also be recursively defined using equational operations. For example, a twinkling light which works forever can be defined as follows:

$$TwinklingLight = on \rightarrow off \rightarrow Twinklinglight$$

2.3.3.2 Semantics

In CSP, the semantics of a process P is defined to be the sequence of events ($traces(P)$) in which the process has engaged up to some moment.

Symbols (Traces)

- $\langle\,\rangle$ The empty trace
- $\langle a\rangle$ a trace with just one element.
- $s \upharpoonright A$ s restricted to A.
- $s \downarrow b$ The amount of times event b appears on trace s.
- $^\frown$ Trace concatenation.
- s_0 the head of s.
- s' The tail of s.

Traces of a Process

$$\begin{array}{lcl}
traces(Stop) & = & \{\langle\,\rangle\} \\
traces(c \rightarrow P) & = & \{\langle\,\rangle\} \cup \{\langle c\rangle^\frown s \mid s \in traces(P)\} \\
traces(P \Box Q) & = & traces(P) \cup traces(Q) \\
traces(\Box S) & = & \bigcup\{traces(P) \mid P \in S\} \\
traces(P \sqcap Q) & = & traces(P) \cup traces(Q) \\
traces(\sqcap S) & = & \bigcup\{traces(P) \mid P \in S\} \\
traces(P|[D]|Q) & = & \bigcup\{s|[D]|t \mid s \in traces(P) \wedge t \in traces(Q)\}
\end{array}$$

Table 2.6: CSP Operational Semantics

2.3.3.3 Verifying Properties in CSP Processes

An specification can be defined as the set of essential requirements that an item or procedure must fulfill. Therefore one can say that a process which satisfies its own specification, guarantees the properties stated in that set of requirements. CSP specifications are given as predicates over traces. Hence we say that a process P satisfies its specification $S(tr)$ if all of its traces satisfy $S(tr)$.

$$P \,\mathbf{sat}\, S(tr) \Leftrightarrow \forall\, tr \in traces(P) \bullet S(tr) \tag{2.3}$$

$P\,\mathbf{sat}\,S(tr)$ can be verified by calculating the traces of P directly from the definitions, establishing that each of them meets the predicate $S(tr)$. In other words, S(tr) is true whenever its variables take values observed from process P. Another way of checking that process P satisfies the specification predicate expressed over traces, is to make use of a set of compositional proof rules, which allow specifications of a process to be deduced from specifications of their components; making use of inference rules with the following structure.

$$\frac{\begin{array}{l} premiss_1 \\ \ldots \\ premiss_n \end{array}}{conclusion}\quad [side - condition]$$

2.3.3.4 Security Protocols in *CSP*

Security protocols work through the interaction of concurrent processes using message-exchanges to communicate with each others. Hence, *CSP* is an adequate tool for modeling all the participants in network and the way in which they are composed as a whole system. Here we will recall the work of Schneider in [Sch96c] for explaining how security matters are modeled in *CSP*.

The architecture of the system consists of a network of nodes (where each node acts as workstation for a particular user) which are able to communicate asynchronously by sending messages to each other by using a medium which acts as a delivery service. The need of security in the system arises from the fact that users in this network do not have control over the medium, and in this way any malicious entity could interfere or intercept the messages transmitted through the common space. This network is modeled in *CSP* in the following way:

$$NETWORK \mathrel{\widehat{=}} (|||_{i \in USER \setminus 0}\, NODE_i)|[trans, rec]|MEDIUM$$

Where all nodes run in a concurrent way interacting with each other through the medium by means of two channels, one by which a node transfers messages to the medium ($trans$) and the other by which receives the data from the medium (rec). Here each user communicates with a particular node, and the nodes are the ones which interact through the medium. The $USER_0$ is omitted because this will be the one representing the enemy. So, as said before, all forms of interference in the network will be modeled by an intruder process $ENEMY = NODE_0$. Now the network is defined as follows:

$$NET \mathrel{\widehat{=}} (|||_{i \in USER \setminus 0}\, NODE_i)|[trans, rec]|MEDIUM|[leak, kill, add]|ENEMY$$

Where the Enemy interacts with the medium by leaking, killing or adding messages.

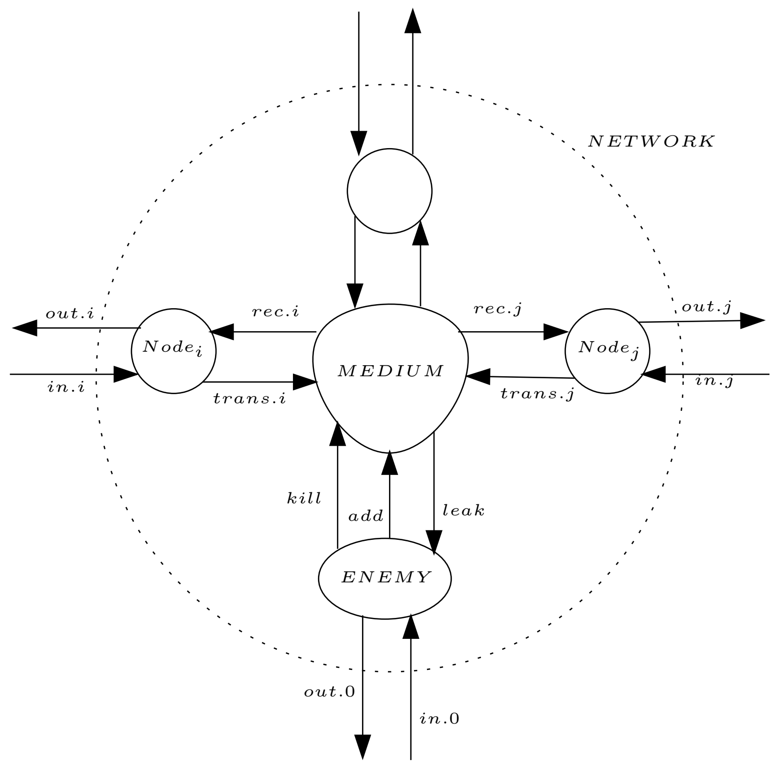


Figure 2.7: Network environment in CSP

2.3.3.5 Modeling and Verifying Security Protocols in *CSP*

Since CSP is not precisely a security process calculi, some general steps have to be followed before making use of its syntax, semantics and proof techniques. As a first step, a particular message space, according to the chosen protocol, has to be specified. For instance, if that specific protocol works with public key encryption, an abstract data type which can capture that cryptographic notion has to be defined. For example a set of $Messages$ is defined st.

$$\begin{array}{lll} MESSAGE & ::= & PLAINTEXT \;|KEY \;|KEY(MESSAGE) \;|MESSAGE.MESSAGE \\ PLAINTEXT & ::= & USER \;|TEXT \;|PLAINTEXT.PLAINTEXT \\ KEY & ::= & PUBLIC\,|\,SECRET \end{array}$$

Where

$$\begin{array}{lll} PUBLIC & = & \{p_i\,|\,i \in USER\} \subseteq KEY \\ SECRET & = & \{s_i\,|\,i \in USER\} \subseteq KEY \\ SECRET & \cap & PUBLIC = \emptyset \end{array}$$

Afterwards, a set of rules concerning the way messages can be generated from existing ones must be defined. These rules, obtained according to the particular message space defined from the protocol to be modeled, will be useful for aiding the proof verification in CSP, acting as basic principles.

Then, the protocol has to be modeled using the syntax provided by this process algebra and the security framework model defined by Schneider. Therefore, each component in the protocol must be defined as a process with its inherent events representing their own behavior. The processes have to be composed together with the medium and the enemy defined lately. This composition is denoted as just one process named as the Network.

Now focusing in the verification phase, several properties of the participants in the protocol have to be formalized, including the medium and the possible intruders that may sabotage the well operation of the network. These properties will combine information about the states and events that have occurred during the run of the protocol. They will be useful later because they will provide us a way of extracting the state of the system from their trace. Before using these specific properties, they have to be verified by means of the rules obtained from the space of messages.

As a last step, a compact specification of the whole network which represents the property wanted to be proved, is modeled as a predicate over traces. Then, the network process is said to be verified, if it satisfies its own specification mentioned before.

The verification mechanism can be achieved by constructing an invariant predicate including the precise reasons why the protocol is expected to work according to the stated properties, verifying it by means of inference rules constructed from the lately established properties and the rules generated from the space of messages, specified for the particular protocol. It is said that the difficulty in finding the adequate invariant for proving a particular property in a protocol, may lead to the discovery of an attack. An example of how lengthy this kind of proofs are, is shown in [Sch96a, Sch96b].

Needham-Schroeder-Lowe protocol in CSP

Here we recall a CSP model of the NSL protocol presented in [Sch96a]. Here channels *trans* and *rec* are of type $USER.USER.MESSAGE$. Where a message $trans.i.j.m$ should be thought as a node i sending a message m with destination j, and $rec.j.i.m$ as a node j receiving a message m from a node i. It can be stated that the value preceded by an ? or by an ! are the input and output values in the event. For example if we say $trans.i!j!m$ it means that the sender (i) is already known and the receiver and message will be the output values j and m respectively. $res.i.j?m$ means that the destination and the source are already known and the only thing the event is awaiting is the message which will be m.

$$\begin{array}{rcl}
USER_a & = & \Box_{i \in USER}\, trans.a!i!p_i(n_a.a) \rightarrow \\
 & & rec.a.i?p_a(n_a.x.i) \rightarrow \\
 & & trans.a!i!p_i(x) \rightarrow Stop \\
USER_b & = & rec.b?a?p_b(y.a) \rightarrow \\
 & & trans.b!a!p_a(y.n_b.b) \rightarrow \\
 & & rec.b.a.p_b(n_b) \rightarrow Stop
\end{array}$$

2.3.4 SPL

SPL is a process calculus designed to model protocols and prove their security properties by means of transitions and event-based semantics. SPL is based on the Dolev-Yao Model, so the assumptions about cryptography and attackers explained in section 2.2.3 are available here. The calculus is operationally defined in terms of configurations containing items of information (messages) which can only increase during evolution, modeling the fact that in an open network an intruder can see and remember any message that was ever in transit.

2.3.4.1 SPL Syntax

The syntactic entities SPL are described below:

- An infinite set N of names denoted by $n, m, ..., A, B, ...$ Names range over *nonces* (randomly generated values, unique from previous choices [Per96]) and agent names.
- Three types of variables: over names (denoted by $x, y, ..., X, Y, ...,$), over keys $(\chi, \chi', \chi_1, ...,)$ and over messages $(\psi, \psi', \psi_1, ...,)$. They could also be expressed as a vector of variables, denoted as $\vec{x}\vec{\chi}\vec{\psi}$ respectively.
- A set of process, denoted by $P, Q, R, ...$

Variables over names	$x, y, ..., X, Y, ...,$
Variables over keys	$\chi, \chi', \chi_1, ...,$
Variables over messages	ψ, ψ', ψ_1
Name expressions	$v \ ::= \ n, A, ... \mid x, X$
Key expressions	$k \ ::= \ Pub(v) \mid Priv(v) \mid Key(\vec{v}) \mid \chi, \chi', ...$
Messages	$M, M' \ ::= \ v \mid k \mid (M, M') \mid \{M\}_k \mid \psi, \psi', ...$
Processes	$p \ ::= \ out\,new(\vec{x})\,M.p \mid in\,pat\vec{x}\vec{\chi}\vec{\psi}\,M.p \mid \Vert_{i\in I}\ P_i \mid !P$

Table 2.7: SPL Syntax

Output	$\langle out\,new(\vec{x})M.p, s, t\rangle \xrightarrow{outnew(\vec{n})M[\vec{n}/\vec{x}]} \langle p[\vec{n}/\vec{x}], s \cup \{\vec{n}\}, t \cup \{M[\vec{n}/\vec{x}]\}\rangle$	if all the names in $\vec{n}$ are distinct and not in s
Input	$\langle in\,pat\,\vec{x}\vec{\chi}\vec{\psi}M.p, s, t\rangle \xrightarrow{inM[\vec{n}/\vec{x},\vec{k}/\vec{\chi},\vec{N}/\vec{\psi}]} \langle p[\vec{n}/\vec{x}, \vec{k}/\vec{\chi}, \vec{N}/\vec{\psi}], s, t\rangle$	if $M[\vec{n}/\vec{x}, \vec{k}/\vec{\chi}, \vec{N}/\vec{\psi}] \in s$
Par. Comp.	$\dfrac{\langle p_j, s, t\rangle \xrightarrow{\alpha} \langle p'_j, s', t'\rangle}{\langle \Vert_{i\in I} P_i, s, t\rangle \xrightarrow{j:\alpha} \langle \Vert_{i\in I} P'_i, s', t'\rangle}$	where $p'_i = p'_j$ for $i = j$, otherwise $p'_i = p_i$

Table 2.8: SPL Transition Semantics

The rest of the elements of SPL syntactic set are defined in Table 2.7, where $Pub(v)$, $Priv(v)$ and $Key(\vec{v})$ denote the generation of public, private and shared keys respectively. We use the vector notation $\vec{s}$ to denote a list of elements, possibly empty, $s_1, s_2, \ldots, s_n$.

2.3.4.2 Intuitive Description and Conventions

Let us now give some intuition and conventions for SPL processes.

The output process $out\,new(\vec{x})\,M.p$ generates a set of fresh distinct names (nonces) $\vec{n} = n_1, n_2, \ldots, n_m$ for the variables $\vec{x} = x_1, x_2 \ldots x_m$. Then it outputs the message $M[\vec{n}/\vec{x}]$ (i.e., M with each x_i replaced with n_i) in the store and resumes as the process $p[\vec{n}/\vec{x}]$. The output process binds the occurrence of the variables $\vec{x}$ in M and p. As an example of a typical output, $p_A = out\,new(\vec{x})\,\{x, A\}_{Pub(B)}.p$ can be viewed as an agent A posting a message with a nonce n and its own identifier A encrypted with the public key of an agent B. We shall write $out\,new(\vec{x})\,M.p$ simply as $out\,M.p$ if the vector $\vec{x}$ is empty.

The input process $in\,pat\,\vec{x}\vec{\chi}\vec{\psi}\,M.p$ is the other binder in SPL binding the occurrences of $\vec{x}\vec{\chi}\vec{\psi}$ in M executing p. As an example of a typical input, $p_B = in\,pat\,x, Z\,\{x, Z\}_{Pub(B)}.p$ can be seen as an agent B waiting for a message of the form $\{x, Z\}$ encrypted with its public key B: If the message of p_A above is in the store, the chosen instantiation for matching the pattern could be the alpha conversion $\{n/x, A/Z\}$, where n matches x and A does the same with Z. When no confusion arises we will sometimes abbreviate $in\,pat\,\vec{x}\vec{\chi}\vec{\psi}\,M.p$ as $in\,M.p$.

Finally, $\|_{i\in I}\; P_i$ denotes the parallel composition of all P_i. For example in $\|_{i\in\{A,B\}}\; P_i$ the processes P_A and P_B above run in parallel so they can communicate. We shall use $!P = \|_{i\in\omega}\; P$ to denote an infinite number of copies of P in parallel. We sometimes write $\|_{i\in\{1,2,\ldots n\}}\; P_i$. to mean $P_1 \parallel P_2 \parallel \ldots \parallel P_n$

The syntactic notions of free variables and closed process/message are defined in an usual way. A variable is *free* in a process/message is has a non-bound occurrence in that process/message. A process/message is said to be *closed* if it has no free variables.

2.3.4.3 Transition Semantics

SPL has a transition semantics over configurations that represents the evolution of processes. A configuration is defined as $\langle p, s, t\rangle$ where p is a closed process term (the process currently executing), s a subset of names $\mathbf{N}$ (the set of nonces generated so far), and t is a subset of variable-free messages (i.e., the store of output messages).

The transitions between configurations are labelled by *actions* which can be input/output and maybe tagged with an index i indicating the parallel component performing the action. Actions are thus given by the syntax $\alpha ::= out\,new(\vec{n})\,M \mid in\,M \mid i : \alpha$. where $\vec{n}$ is as a set of names, i as an index and M a closed message.

Intuitively a transition $\langle p, s, t\rangle \xrightarrow{\alpha} \langle p', s', t'\rangle$ says that by executing α the process p with s and t evolves into p' with s' and t'. The new set of messages t' contains those in t since output messages are meant to be read but not removed by the input processes. The rules in Table 2.8 define the transitions between configurations. The rules are easily seen to realize the intuitive behavior of processes given in the previous section.

Nevertheless, SPL also provides an *event based semantics*, where events of the protocol and their dependencies are made more explicit. This is advantageous because events and their pre and post-conditions form a Petri-net, so-called SPL nets.

2.3.4.4 Event-Based Semantics

Although transition semantics provide an appropriate method to show the behavior of configurations, these are not enough to show dependencies between events, or to support typical proof techniques based on maintenance of invariants along the trace of the protocols. To do so, SPL presents an additional semantics based in events that allow to explicit protocol events and their dependencies in a concrete way.

SPL event-based semantics are strictly related to persistent Petri nets, so called *SPL-nets* [Cra03] defining events in the way they affect conditions. The reader may find full details about Petri Nets and all the elements of a SPL-Nets in Appendix A and [Cra03], below we just recall some basic notions.

Description of Events in SPL In the event-based semantics of SPL, conditions take an important place as they represent some form of local state. There are three kinds of conditions: *control*, *output* and *name* conditions (denoted by C, O and N, respectively). C-conditions includes input and output processes, possibly tagged by an index. O-conditions are the only persistent conditions in SPL-nets and consists of closed messages output on network. Finally, N-conditions denotes basically the set of names $\mathbf{N}$ being used for a transition. In order to denote pre and post conditions between events, let ${}^{\cdot}e = \{{}^{c}e, {}^{o}e, {}^{n}e\}$ denote the set of control, name and output preconditions, and $e^{\cdot} = \{e^{c}, e^{o}, e^{n}\}$ the equivalent set of postconditions. An SPL event e is a tuple $e = ({}^{\cdot}e, e^{\cdot})$ of the preconditions and postconditions of e and each event e is associated with a unique action $act(e)$. Figure 2.8 gives the general form of an SPL event. The exact definition of each element of the events can be found in [Cra03].

To illustrate the elements of the event semantics, consider a simple output event $e = (\mathbf{Out}(out\ new\vec{x}M); \vec{n})$, where $\vec{n} = n_1 \dots n_t$ are distinct names to match with the variables $\vec{x} = x_1 \dots x_t$. The action $act(e)$ corresponding to this event is the output action $out\, new\vec{n}M[\vec{n}/\vec{x}]$. Conditions related with this event are:

$$\begin{array}{lll}
{}^{c}e = \langle out\, new(\vec{x}).M.p, a\rangle & {}^{o}e = \emptyset & {}^{n}e = \emptyset \\
e^{c} = \langle Ic(p[\vec{n}/\vec{x}])\rangle & e^{o} = \{M[\vec{n}/\vec{x}]\} & e^{n} = \{n_1, \dots n_t\}
\end{array}$$

Where $Ic(p)$ stands for the initial control conditions of a closed process p: The set $Ic(p)$ is defined inductively as $Ic(X) = \{X\}$is X is an input or an output process, otherwise $Ic(\|_{i\in I}\, P_i) = \bigcup_{i\in I}\{i : c \mid c \in Ic(P_i)\}$

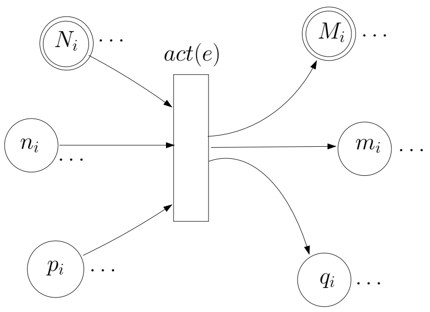


Figure 2.8: Events and transitions of SPL event based semantics: p_i and q_i denote control conditions, n_i and m_i name conditions and N_i, M_i output conditions. Double circled conditions denote persistent events.

2.3.4.5 Relating Transition and Event Based Semantics

Transition and event based semantics are strongly related in SPL by the following theorem from [Cra03]. The reduction $M \xrightarrow{e} M'$ where e is an event and M and M' are markings in the SPL-net is defined in the Appendix following the token game in Persistent Petri Nets (see Appendix A).

Theorem 1. i) *If* $\langle p, s, t\rangle \xrightarrow{\alpha} \langle p', s', t'\rangle$, *then for some event* e *with* $act(e) = \alpha$, $Ic(p) \cup s \cup t \xrightarrow{e} Ic(p') \cup s' \cup t'$ in the SPL-net.

ii) *If* $Ic(p) \cup s \cup t \xrightarrow{e}$ *M' in the* **SPL**-*net, then for some closed process term* p', *for some* $s' \subseteq N$ *and* $t' \in O$, $\langle p, s, t\rangle \xrightarrow{act(e)} \langle p', s', t'\rangle$ *and* M' $= Ic(p') \cup s' \cup t'$.

Justified in the theorem above, the following notation will be used: Let e be an event, p be a closed process, $s \subseteq N$, and $t \subseteq O$. We write $\langle p, s, t\rangle \xrightarrow{e} \langle p', s', t'\rangle$ iff $Ic(p) \cup s \cup t \xrightarrow{e} Ic(p') \cup s' \cup t'$ in the **SPL**-net.

2.3.4.6 Events of a Process

Each process has its own related events, and for a particular closed process term p, the set of its related events $Ev(p)$ is defined by induction on size, in the following way:

$Ev(out\, new\, \vec{x} M.p) \quad = \quad \{\, \mathbf{Out}\ (out\, new\, \vec{x} M.p;\, \vec{n})\} \cup \bigcup \{Ev(p[\vec{n}/\vec{x}])\}$

Where $\vec{n}$ are distinct names

$Ev(in\, pat\, \vec{x}\vec{\chi}\vec{\psi} M.p) \quad = \quad \{\mathbf{In}(in\, pat\, \vec{x}\vec{\chi}\vec{\psi} M.p;\, \vec{n}, \vec{k}, \vec{L})\} \cup \bigcup \{Ev(p[\vec{n}/\vec{x}, \vec{k}/\vec{\chi}, \vec{L}/\vec{\psi}])\}$

Where $\vec{n}$ names, $\vec{k}$ are keys, and $\vec{L}$ are closed messages

$Ev(\|_{i \in I} p_i) \quad = \quad \bigcup_{i \in I} i : Ev(p_i)$

where, E is a set, and $i : E$ denotes the set $\{i : e \mid e \in E\}$.

2.3.4.7 General Proof principles

Verifying security properties in SPL is not as tedious as in other calculi since, its inherent proof techniques are based on its own operational principles. In other words, SPL uses its event based semantics to derive some general proof principles, which capture the notion of dependency between events in a protocol run. These principles, are of the essence of SPL's proof techniques but they are not the only concepts used for aiding the properties' verification. The proofs are simplified by a result of the occurrence of the spy events in the protocol run. The result is based on the notion of surroundings of a message inside another. These ideas inherent from the calculus are the ones used to verify or contradict the fulfillment of any security property in a protocol run.

From the net semantics we can derive several principles useful in proving authentication and secrecy of security protocols. Write $M \sqsubseteq M'$ to mean message M is a subexpression of message M', i.e., $\sqsubseteq$ is the smallest binary relation on messages st:

$$\begin{array}{lll} M \sqsubseteq M & & \\ M \sqsubseteq N & \Rightarrow & M \sqsubseteq N, N' \text{ and } M \sqsubseteq N', N \\ M \sqsubseteq N & \Rightarrow & M \sqsubseteq \{N\}_k \end{array}$$

where M, N, N' are messages and k is a key expression. We also write $M \sqsubset t$ iff $\exists M'.M \sqsubset M' \wedge M' \in t$, for a set of messages t.

Below we present a set of general proof principles strongly based on the work done by Federico Crazzolara in [Cra03].

Definition 9 (Well-foundedness)**.** Given a property P on configurations, and $P(p_0, s_0, r_0)$ represents that configuration $\langle p_0, r_0, s_0 \rangle$ holds property P, if a run $\langle p_0, s_0, t_0 \rangle \xrightarrow{e_1} ... \xrightarrow{e_r} \langle p_r, s_r, t_r \rangle \xrightarrow{e_{r+1}} ...$, contains configurations st $P(p_0, s_0, t_0)$ and $\neg P(p_j, s_j, t_j)$, then there is an event e_h, $0 < h \leq j$, st. $P(p_i, s_i, t_i)$ for all $i < h$ and $\neg P(p_h, s_h, t_h)$.

We say that a name $m \in N$ is *fresh* on an event e if $m \in e^n$ and we write $Fresh(m, e)$

Definition 10 (Freshness)**.** Within a run

$$\langle p_0, s_0, t_0 \rangle \xrightarrow{e_1} ... \xrightarrow{e_r} \langle p_r, s_r, t_r \rangle \xrightarrow{e_{r+1}} ...,$$

the following properties hold:

1. If $n \in s_i$ then either $n \in s_0$ or there is a previous event e_j st $Fresh(n, e_j)$.
2. Given a name n there is at most one event e_i st $Fresh(n, e_i)$.
3. If $Fresh(n, e_i)$ then for all $j < i$ the name n does not appear in $\langle p_j, s_j, t_j \rangle$.

Definition 11 (Control Precedence)**.** Within a run

$$\langle p_0, s_0, t_0 \rangle \xrightarrow{e_1} ... \xrightarrow{e_r} \langle p_r, s_r, t_r \rangle \xrightarrow{e_{r+1}} ...,$$

if $b \in {}^c e_i$ either $b \in Ic(p_0)$ or there is an earlier event e_j, $j < i$, st $b \in e_j^o$.

Definition 12 (Output-input Precedence.)**.** Within a run

$$\langle p_0, s_0, t_0\rangle \xrightarrow{e_1} ... \xrightarrow{e_r} \langle p_r, s_r, t_r\rangle \xrightarrow{e_{r+1}} ...,$$

if $M \in {}^{o}e_i$, then either $M \in t_0$ or there is an earlier event e_j, $j < i$, st. $M \in e_j^o$

Definition 13 (Output Principle.)**.** Within a run

$$\langle p_0, s_0, t_0\rangle \xrightarrow{e_1} ... \xrightarrow{e_r} \langle p_r, s_r, t_r\rangle \xrightarrow{e_{r+1}} ...,$$

According to the message persistence in SPL, $\forall e_v$ in a run, $e_v^o - e_{v-1}^o$ are the new messages generated by event e_v.

2.3.4.8 Message Surroundings

Given a pair of messages M and N the surroundings of N in M are the smallest submessages of M containing N under one level of encryption. So for example the surroundings of $Key(A)$ in

$$(A, \{B, Key(A)\}_k, \{Key(A)\}_{k'})$$

are $\{B, Key(A)\}_k$ and $\{Key(A)\}_{k'}$. If N is a submessage of M but does not appear under encryption in M then we take the surroundings of N in M to be N itself.
For example the surroundings of $Key(A)$ in

$$(A, \{B, Key(A)\}_k, Key(A))$$

are $\{B, Key(A)\}_k$ and $Key(A)$.

Let M and N be two messages. Define $\sigma(N, M)$ the surroundings of N in M inductively as follows:

$$\begin{array}{lll}
\sigma(N,v) & = & \begin{cases} \{v\} & \text{if } N = v \\ \emptyset & otherwise \end{cases} \\
\sigma(N,k) & = & \begin{cases} \{k\} & \text{if } N = k \\ \emptyset & otherwise \end{cases} \\
\sigma(N,(M,M')) & = & \begin{cases} \{(M,M')\} & \text{if } N = M, M' \\ \sigma(N,M) \cup \sigma(N,M') & otherwise \end{cases} \\
\sigma(N,\{M\}_k) & = & \begin{cases} \{\{M\}_k\} & \text{if } N \in \sigma(N,M) \text{ or } N = \{M\}_k \\ \sigma(N,M) & otherwise \end{cases} \\
\sigma(N,\psi) & = & \begin{cases} \{\psi\} & \text{if } N = \psi \\ \emptyset & otherwise \end{cases}
\end{array}$$

2.3.4.9 Proving Security Properties with SPL

There are some general steps which must be followed in order to verify security properties under this framework.

Any security property wanted to be verified must be modelled in a formal way. This can be done in a very intuitive way, by means of the notions of message surroundings, which capture the most important concepts needed for representing security predicates. Afterwards, in order to fulfill the already formalized property, every event in the protocol must be verified. An adequate method for proving these properties over such different events is the contradiction mechanism, by which one states a simple supposition, such as the existence of an event in which the property is not achieved, and by means of the event dependency presented in the SPL language and the proof principles mentioned before, one tries to find that the event which must exist in order to broke the property never happens along the protocol run, as can be seen in chapters 3 and 4.

2.4 Discussion and Calculus Selection

Process Calculi can be seen as an accurate set of models that allows to express the behavior of communication protocols from an operational and intuitive way. A particularity of the process calculi studied so far is the inclusion of elements which allow covering several security issues, such as:

1. Cryptographic primitives.
2. Fresh name generation,
3. Execution environments to formally verify security protocols and ways to model attackers.
4. Well founded reasoning techniques specially devoted to cover important aspects in security.

By means of these elements we will establish a comparison between a representative set of process calculi for security, in order to select the one best suited for security analysis with respect to the previous criteria. As can be seen in table 2.9.

	π Calculus	Spi Calculus	CSP	SPL
1	Private Channels	Available	Available	Available
2	Available	Available	Not Available	Available
3	Available / Linear	Available / Linear	Available / Linear	Available / Monotonic
4	Not Available	Available	Available	Available

Table 2.9: Comparative analysis between process calculi concerned to security

According to this items, the π Calculus, and a particular extension named the Spi calculus, define agents involved in communication tasks as processes interacting over a set of channels, establishing scope rules and equivalences to determine when a message can be leaked by an attacker. The Spi calculus goes further and adds a set of primitives in the operational semantics representing the operation of cryptographic datatypes in concurrent comunication like nonce generation and encryption, guaranteeing security properties by means of holding a set of equivalence between Spi processes. However, although the Spi calculus is well equipped with a set of reduction rules that aids the analysis of equivalences [AG97a]; one needs to relate every model to a sort of "magical", correct and secure implementation that does not disclose any message received, making the proofs rather complicated and far from intuitive reasoning.

The abstract level of specification turns CSP as an optimal option to model different types of protocols, such as those concerned to security. In fact, several models are defined to extent the algebra with datatypes and properties for security [Sch96c, RSG$^+$01]. However, a typical proof includes the revision of the model from the scratch, defining particular environments, attacker abilities, deduction rules and invariants for every protocol. This approach makes a proof very tedious and lengthy. Another disadvantage is the lack of replication methods for process or messages, something determinant to express the persistence of messages in the network and infinite behavior of process in communication, such as servers and P2P systems.

SPL provides a different approach for process calculi. It is strongly based on event semantics, which represents protocol evolution in a clear and intuitive manner, includes primitives to show cryptography operations, is based on a persistent model of network where each of the messages sent is maintained for an unlimited period of time, representing the power of an attacker to infinitely collect information from the network, and supply clear proof techniques where a property is ensured if an event that violates the preconditions is found. This characteristic turns to an ideal model well suited to security analysis in concurrent and infinite systems, which we will show in chapters 3 and 4.

2.5 Summary

Along this chapter, we present a general overview of the way in which communication has evolved through time, and the way in which security properties have arose as crucial concepts when analyzing these kind of systems. We begin with a general idea of communication used in daily live, passing through more sophisticated concepts by which this kind of systems are modeled in an informal or formal way, and up to theories, algebras or process calculi by which several characteristics such as security are verified. We initiate with an introduction to communication as a general concept, continuing our description path by presenting a general overview of the first approaches developed for studying communication systems and we finish with the notion of process calculi, where we present those only concerned to communication concurrent systems, such as CCS or the π-calculus, and then, those which include some notions about security, like the *Spi*-calculus, CSP, and SPL.

We give out a deeper presentation of some important process calculi concerned to security, where we present their basic principles, among with a general description of their syntax and their operational semantics. Showing their proof principles by which these languages are based, as well as some examples of simple protocols modeled using those languages.

After a brief discussion of the chapter, we choose the most appropriate security process calculus, according to concurrency needs, expressive power, operational semantics and reasoning techniques used to verify security protocols.

3 MUTE Protocol: Secrecy over P2P systems

Peer to peer (P2P) systems rely on the concept of employing several distributed resources, such as computer power, data or network bandwidth, to perform a critical function in a decentralized way instead of concentrating in one central entity. Examples of tasks suitable for this computing scheme include: distributed computing, data content sharing, communication and collaborative systems [Ese02, BS04, GK03, BMWZ05, Rip01].

P2P networks lack of a clear notion of clients or servers. All participants in these networks are denoted as simply peers which, according to the circumstances may work as clients, or servers. In that way, there is no need of having a central entity by which a client requests and receives any type of information; instead, the data flow may come from any peer inside the network, since any peer can respond acting as a server. In this way, there is a much lower cost of ownership or sharing, since there is a use of an existent infrastructure, and there is also an elimination and reduction of maintenance costs, by distributing jobs through all the participants in the net.

Protocols for P2P systems are used to share private information between peers, which usually involves security risks. Currently these systems are dramatically receiving attention in research, development and investment. They had become a major force in the nowadays computing world because of its huge amount of benefits, such as its architecture cost, scalability, viability, and resource aggregation of distributed management resources.

The P2P protocols used in various tools should maintain a number of important properties to guarantee their well functioning. One of the most important properties in P2P protocols are those concerned to security. Properties like secrecy and non-traceability have been studied in the literature in order to overcome security risks [MKL$^+$02]. Secrecy is considered important, since we may want to keep secret from an entity outside the P2P group, the messages transmitted and managed between the components within the network. Obviously, in some groups a malicious outsider may easily become an insider by signing up as a peer. However, one can imagine situations when becoming a peer requires to show that the potential peer can be trusted, or to provide certain information the outsider is not capable or willing to give.

Despite the popularity of this kind of systems, the importance of maintaining security matters within them and the existence of different calculi to reason about protocols, to the best of our knowledge, little has been done in modeling P2P protocols using process calculi.

In this chapter we are going to explore the security issues of a P2P system by modeling a protocol widely used in these kind of systems known as MUTE [RR05]. The general structure we shall follow for modeling MUTE, will be the next: In our first part, we extract a formal model directly from the implementation code. Then, using the SPL formalism along with its compositional power, we establish the formal specification of the MUTE protocol searching phase, modeling their components as a set of processes which work together to achieve the main goal of the protocol. Finally we use the proof techniques of SPL to prove a secrecy property for the messages in the network with respect to a malicious outsider. In the second part we make some modifications into the original MUTE protocol, in order to guarantee a much stronger secrecy property. By means of the SPL language we specify this new protocol. Then, using the language proof techniques, we verify the secrecy property behind a saboteur inside the network.

3.1 Protocol Description

MUTE is a P2P tool for sharing and transmitting resources in a highly dynamic distributed network [RR05]. It is based on a particular searching protocol, which claims to guarantee an anonymous way of communicating data in a secure way through the P2P network. In spite of being a real life protocol, MUTE has only been informally described. Following our original approach, we shall use SPL to give a formal specification of the MUTE protocol.

The MUTE protocol works in a P2P network as a tool to communicate requests of keywords through the net, so that an specific file can be found and then received [RR05]. This protocol is composed of two main phases: searching and routing parts. We will focus directly in its first phase, since it is the most related to the security concerns related to our work.

This protocol aims to provide an easy and effective search while protecting the privacy of the participants involved. It is inspired in the behavior of ants in the search for food. Despite of ants having a simpler brain than humans, they do have a collectively more intelligent route finding technique than human beings. In principle, ants search for food in a very simple way, they just walk randomly until reaching their target. The crucial point is that each ant leaves a trail of pheromones as it searches for food. In that way they just have to follow back their own trail to reach their home. The essential fact in this behavior is concerned to the help of each ant to the rest of the anthill by showing them the shortest path, even though they do not have a special way of telling others which one is the best. The notion of pheromones works again as the solution for this problem. Ants which go back home following their own trial, leave more pheromones along their way giving a much stronger scent to the path and attracting more ants in that way [DS04].

This notion of routing in ants colonies, plays a very important role in some P2P protocols such as MUTE. The analogy between ants route finding technique and P2P protocols is accomplished by representing each ant as a node of a network, files requested as food, and pheromones as traces. In this way, one of the key properties of this model is the inherent anonymity of the protocol, because, as the ants that ignore the shortest path between the food and the anthill, peers are unaware of the overall environment layout and MUTE messages must be directed through the network using only local hints [1].

Since the MUTE protocol claims to have anonymous users, none of the nodes in the P2P network knows where to find a particular recipient. Each node in the MUTE network contains direct connections to other nodes in the network in order to achieve its desired search. This nodes are called "neighbors" and through these, messages are secretly passed, either as a request or as an answer, in such a way that no agent outside the peer to peer network could manage to understand any of these data. Despite anonymity being essential on this protocol, secrecy is also one of its main goals, since transmitted messages along the network involve information only concerned to the ones sharing the resources and must not be revealed to the outside world.

3.2 Dolev-Yao Representation

In spite of being already implemented and used as a tool for downloading and sharing files, to our knowledge MUTE has not yet been formally specified. Part of our work consists in abstracting from the code elements that have an impact in security.

Definition 14 (Sets in Mute)**.** Let $Files$ be the set of all files in the P2P network and $Files(A)$ the set of files belonging to peer A. Let $Keywords$ be the set of keywords associated to the files $Files$, $Keywords(A)$ the keywords associated to the peer A and $Keys$ the relation $Files : Keywords$, representing the keywords associated to a particular file. Let $Headers$ be the set of headers of files, which is associated to $Files$, $Headers(A)$ the set directly related to $Files(A)$, such that each header which belongs to $Headers(A)$ will be associated to a unique file belonging to $Files(A)$.

Definition 15 (P2P Network model)**.** We shall describe a P2P network as an undirected graph G whose nodes represent the peers and whose edges mean the direct connections among them. We use $Peers(G)$ to denote the set of all nodes in G. Given a node $X \in Peers(G)$, Let $ngb(X)$ be the set of immediate neighbors of X. We use the Dolev-Yao notation $X \longrightarrow Y : M$ stating that X sends a message M to Y.

For Example, consider a P2P network G with $A, B \in Peers(G)$. Suppose that A initiates the protocol by broadcasting a request to all its neighbors in order to find a particular answer, and B is the agent which has the desired answer that A is searching for, deciding to send a response. In this case, B can be any node inside the network with the desired file on its store. A requests for a particular file he wishes to download, sending the request to the network by broadcasting it to his neighbors. This request includes a keyword $kw \in Keywords$, which will match the desired file, and a nonce N which will act as the request identifier. Along the searching path an unknown amount of peers will forward the request until B is reached, the

[1] Abstracting from the MUTE website, available at [RR05]

peer with the correct file st. $\exists f \in Files(B)$ and $kw \in Keys(f)$. Then, B sends its response by means of the header of the file RES, among with the identifier N and a new name M generated by it to recognize the message as an answer. This is done again by broadcasting the message through a series of forward steps, until reaching the actual sender A. Figure 3.1 give a representation of the above description using Dolev-Yao notation [DY81].

$$\begin{array}{lll}
A \longrightarrow X: & (\{N, Kw\}_{key(A,X)}, A, X) & \text{for } X \in ngb(A) \\
X \longrightarrow Y: & (\{N, Kw\}_{key(X,Y)}, X, Y) & \text{for } Y \in ngb(X) \\
\vdots & & \\
Z \longrightarrow B: & (\{N, Kw\}_{key(Z,B)}, Z, B) & \\
B \longrightarrow X': & (\{N, RES, M\}_{key(A,X')}, A, X') & \text{for } X' \in ngb(B) \\
X' \longrightarrow Y': & (\{N, RES, M\}_{key(X',Y')}, X', Y') & \text{for } Y' \in ngb(X) \\
\vdots & & \\
Z' \longrightarrow A: & (\{N, RES, M\}_{key(Z',A)}, Z', A) &
\end{array}$$

Figure 3.1: Dolev-Yao Model of the MUTE protocol

Here X, Y, Z are variables which represent the peers which forward the message along the path going from agent A to B. This process may continue until the target is reached. Meanwhile the X', Y', Z' variables will represent the peers which will forward the answer from B to A. This process may be repeated several times as well.

3.3 An SPL Specification of MUTE

We use the core of the MUTE protocol in order to establish some security properties, assuming a previous connection stage between neighbors. The phases that we shall consider are the ones involving the transmission of a keyword, the response message and the keys, and the sub-messages including plaintexts. We assume that the symmetric keys $key(A, B) = key(B, A)$. The formal model is presented in Figure 3.2, introducing a notation $(\|_{i\in I}\ P_i) \,.\, R$ as a valid construction easily encoded, where R could be any kind of process in the language, as can be seen below:

$(\|_{i\in I}\ P_i).R = out\, new(\vec{T})\, x\, .(\|_{i\in I}\ P_i.out\, \{x\}_{Pub(T_i)}) \parallel (in\, \{x\}_{Pub(T_i)})^I.R$
Where:
$(in\{x\}_{Pub(T_i)})^I = in\, \{x\}_{Pub(T_1)}.\, in\{x\}_{Pub(T_2)}....in\{x\}_{Pub(T_I)}$

In MUTE there are essentially three main roles which describe the behavior of the peers in the whole process: The initiator, the intermediator and the responder. The initiator is the agent that starts the protocol by means of a request, the intermediator the one that forwards the message request and the responder, the peer which has the actual answer for the request and sends back the response in order to answer the initiator's query. Any peer inside the network can execute any of these roles.

The composition of three processes representing the main roles in MUTE give form to the following model:

$$\begin{array}{lcl}
Init(A) & \equiv & (\|_{B\,\in\,ngb(A)}\ out\,new(n)(\{n, Kw\}_{Key(A,B)}, A, B))\,. \\
 & & (\|_{Y\,\in\,ngb(A)}\ in\,(\{n, res, m\}_{key(Y,A)}, Y, A)) \\[1em]
Interm(A) & \equiv & !\,(\|_{Y\,\in\,ngb(A)}\ in(\{M\}_{key(Y,A)}, Y, A)\,.\,\|_{B\,\in\,ngb(A)-\{Y\}}\ out\,(\{M\}_{key(A,B)}, A, B)) \\[1em]
Resp(A) & \equiv & \|_{Y\,\in\,ngb(A)\,,\,kw\,\in\,Keys(Files(A))}\ in(\{x, Kw\}_{Key(Y,A)}, Y, A)\,. \\
 & & (\|_{B\,\in\,ngb(A)} out\,new(m)(\{x, res, m\}_{key(A,B)}, A, B)) \\[1em]
Node(A) & \equiv & Init(A)\ \|\ Interm(A)\ \|\ Resp(A) \\
SecureMUTE & \equiv & \|_{A\,\in\,Peers(G)} Node(A)
\end{array}$$

Figure 3.2: MUTE specification on SPL

We assume that the topology of the net has already been established. A typical execution of the protocol starts with the initiator searching for an own keyword. This agent broadcasts the desired keyword to all its neighbors ($\|_{B\in ngb(A)} out\,new(n)\,(\{n, Kw\}_{Key(A,B)}, A, B)$). Its neighbors receive the message and check if the keyword matches one of their files ($\|_{Y\,\in\,ngb(A)\,,\,kw\,\in\,Keys(Files(A))}\ in\,(\{x, Kw\}_{Key(Y,A)}, Y, A))$. If at least one of the neighbors have the requested keyword, then such a neighbor will broadcast a response message $\|_{B\in ngb(A)} out\,new(m)$ $(\{x, res, m\}_{key(A,B)}, A, B)$, such that eventually the peer searching for the keyword will get this response $in(\{n, res, m\}_{key(Y,A)}, Y, A)$ and understands it as an answer to its request. The message will be forwarded by all the agents until it reaches its destiny ($\|_{B\,\in\,ngb(A)-\{Y\}}$ $out\,(\{M\}_{key(A,B)},\ A, B)$). Otherwise, if the keyword does not match any file of the agent, then it will broadcast it to its neighbors asking them for the same keyword ($\|_{B\,\in\,ngb(A)-\{Y\}}$ $out\,(\{M\}_{key(A,B)},\ A, B)$). The choice of having or not the right file is modeled in a non-deterministic way. This model abstracts away from issues such as the search for the best path, since it has no impact in secrecy.

3.4 Events

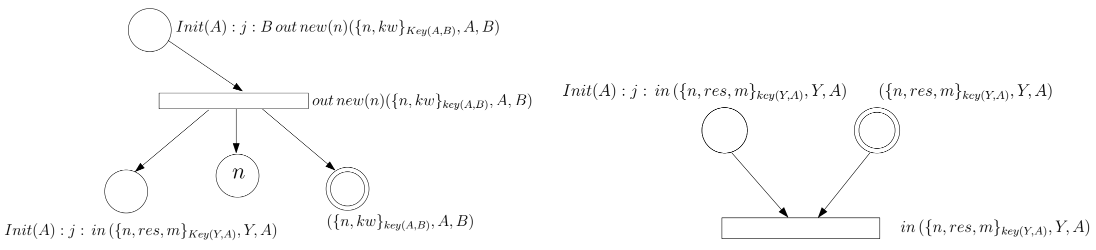
Definition 16 (Events in MUTE)**.** The event e_w is an event in the set

$$Ev(\text{MUTE}) = Init : Ev(p_{Init}) \cup Interm : Ev(p_{Interm}) \cup Resp : Ev(p_{Resp}) \cup Spy : Ev(p_{Spy})$$

Where the events are graphically represented in figures 3.3, 3.4 and 3.5.

3.4.1 Initiator Events

The initiator events indicate the behavior of process $Init(A)$. This process can be splitted in two main sub-processes: an output that generates a new name n and a request message $(\{n, kw\}_{Key(A,B)}, A, B)$ over the store (figure 3.3(a)), and an input process that receives the answer message $(\{n, res, m\}_{Key(A,B)}, A, B)$ via an input action $in\,(\{n, res, m\}_{Key(A,B)}, A, B)$, as can be seen in figure 3.3(b).

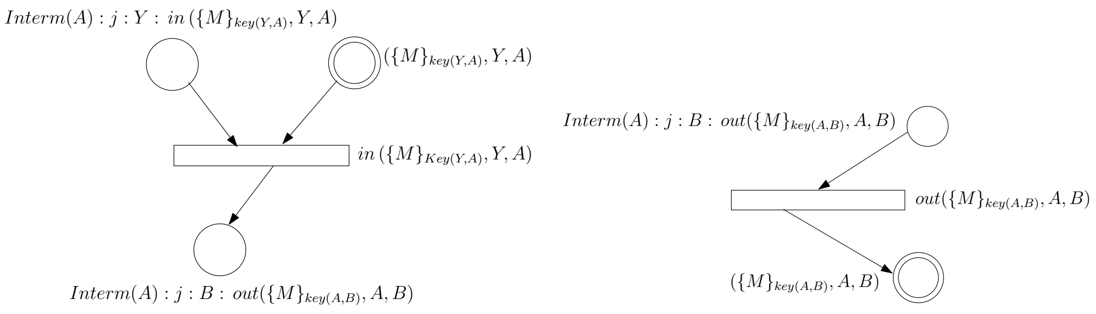

$Init(A) : j : B\; out\; new(n)(\{n, kw\}_{Key(A,B)}, A, B)$
$out\; new(n)(\{n, kw\}_{key(A,B)}, A, B)$
n
$Init(A) : j :\; in\,(\{n, res, m\}_{Key(Y,A)}, Y, A)$
$(\{n, kw\}_{key(A,B)}, A, B)$
$Init(A) : j :\; in\,(\{n, res, m\}_{key(Y,A)}, Y, A)$
$(\{n, res, m\}_{key(Y,A)}, Y, A)$
$in\,(\{n, res, m\}_{key(Y,A)}, Y, A)$

(a) Initiator Output (b) Initiator Input

Figure 3.3: Initiator Events

3.4.2 Intermediator Events

Each agent acting as an intermediator has to forward the received messages. The figure 3.4(a) illustrates the event in which the intermediator receives the message $(\{M\}_{Key(Y,A)}, Y, A)$ via an input action $in\,(\{M\}_{Key(Y,A)}, Y, A)$. The composition of a second subprocess (figure 3.4(b)) completes the intermeditator behavior, forwarding received messages M to one of the neighbors by means of an output $out\,(\{M\}_{Key(A,B)}, A, B)$.

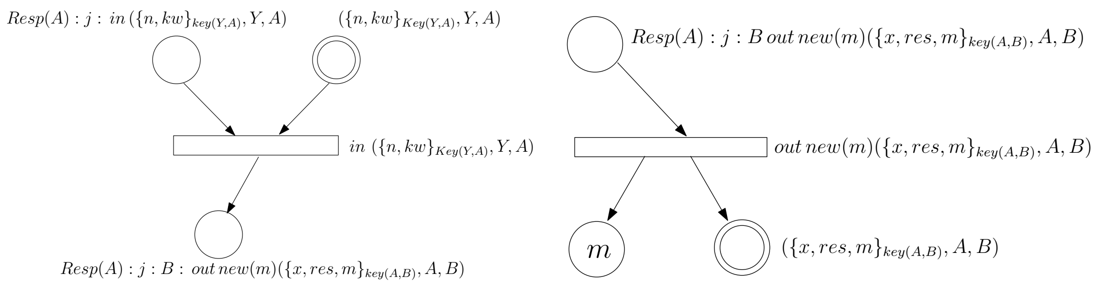

$Interm(A) : j : Y :\; in\,(\{M\}_{key(Y,A)}, Y, A)$
$(\{M\}_{key(Y,A)}, Y, A)$
$in\,(\{M\}_{Key(Y,A)}, Y, A)$
$Interm(A) : j : B :\; out(\{M\}_{key(A,B)}, A, B)$
$Interm(A) : j : B :\; out(\{M\}_{key(A,B)}, A, B)$
$out(\{M\}_{key(A,B)}, A, B)$
$(\{M\}_{key(A,B)}, A, B)$

(a) Intermediator Input (b) Intermediator Output

Figure 3.4: Intermediator Events

3.4.3 Responder Events

The responder events indicate the way in which an agent acting as a responder must behave. A responder agent is basically composed by two processes: An initial input (figure 3.5(a)) that awaits for a message request $(\{n, kw\}_{Key(Y,A)}, Y, A)$, and a subsequent output of the answer $(\{n, res, m\}_{Key(A,B)}, A, B)$ via an output action $out\,(\{n, res, m\}_{Key(A,B)}, A, B)$, with a new name m (figure 3.5(b)).



$Resp(A) : j : in(\{n, kw\}_{key(Y,A)}, Y, A)$
$(\{n, kw\}_{Key(Y,A)}, Y, A)$
$in\ (\{n, kw\}_{Key(Y,A)}, Y, A)$
$Resp(A) : j : B : out\, new(m)(\{x, res, m\}_{key(A,B)}, A, B)$
$Resp(A) : j : B\, out\, new(m)(\{x, res, m\}_{key(A,B)}, A, B)$
$out\, new(m)(\{x, res, m\}_{key(A,B)}, A, B)$
m
$(\{x, res, m\}_{key(A,B)}, A, B)$

(a) Responder Input (b) Responder Input

Figure 3.5: Responder Events

3.5 MUTE Secrecy Proofs behind an Outsider Spy

Here we will establish the secrecy of MUTE for a Spy outside the P2P network

3.5.1 Definition of the Spy

Using a well studied model of spy [Cra03], a possible attacker over the network is presented in table 3.1

Compose different messages into a single tuple	$Spy_1 \equiv in\psi_1.in\,\psi_2.out\,\psi_1, \psi_2$
Decompose a compose message into more components	$Spy_2 \equiv in\,\psi_1, \psi_2.out\,\psi_1.out\,\psi_2$
Encrypt any message with the keys that are available	$Spy_3 \equiv in\,x.in\,\psi.out\,\{\psi\}_{Pub(x)}$ $Spy_4 \equiv in\,Key(x, y).in\,\psi.out\,\{\psi\}_{Key(x,y)}$
Decrypt messages with available keys	$Spy_5 \equiv in\,Priv(x).in\,\{\psi\}_{Pub(x)}.out\,\psi$ $Spy_6 \equiv in\,Key(x, y).in\,\{\psi\}_{Key(x,y)}.out\,\psi$
Sign with available keys	$Spy_7 \equiv Priv(x).in\,\psi.out\,\{\psi\}_{Priv(x)}$
Verify signatures with available keys	$Spy_8 \equiv in\,x.in\,\{\psi\}_{Priv(x)}.out\,\psi$
Create new random values	$Spy_9 \equiv out\,new(\vec{n})\vec{n}$

Table 3.1: SPL spy model

Finally, the complete Spy is a parallel composition of the Spy_i processes:

$$Spy \equiv \|_{i\in\{1...9\}} Spy_i \tag{3.1}$$

In this way, the complete protocol includes the specification of MUTE, $SecureMute$ in Figure 3.2, in parallel with the Spy:

$$MUTE \equiv SecureMUTE \| !Spy \tag{3.2}$$

To Analyze secrecy of a given protocol in SPL, one considers arbitrary runs of the protocol.

Definition 17 (Run of a Protocol)**.** A run of a process $p = p_0$ is a sequence

$$\langle p_0, s_0, t_0 \rangle \xrightarrow{e_1} \cdots \xrightarrow{e_w} \langle p_w, s_w, t_w \rangle \xrightarrow{e_{w+1}} \cdots$$

We shall use in the theorems a binary relation $\sqsubset$ between messages, defined in 2.3.4.7.

3.5.2 Secrecy Proofs in MUTE

To guarantee an important security property such as secrecy behind an outsider spy over distributed environments such as the one presented in MUTE, we must follow a series of steps which include several individual proofs before ensuring the property for the whole protocol.

As the first step we must verify that the shared keys used by peers inside the network for encrypting messages sent between them, are never leaked during message transmissions. This is actually a very important property since it ensures that information encrypted with these keys, is never understood by saboteurs outside the P2P network.

Then, assuming that those keys are never leaked, we can verify the secrecy properties for the two kinds of messages transmitted along the protocol, the answer and the request. A very straightforward way of verifying that those messages are kept as a secret is to present a stronger property stating that answers and requests always appear inside messages encrypted with the shared keys, and since we know that messages encrypted with these keys can never be decrypted by outsiders, therefore the secrecy property for answers and requests is fulfilled. In order to verify this property, each output event occurring in the protocol must be verified, to ensure that there is no message where answers or requests appear in non ciphered messages. Then, if the secrecy property for answers and requests is achieved in a protocol run, we can state that the whole protocol fulfills the secrecy property.

3.5.2.1 Secrecy property for shared keys

This theorem for the MUTE protocol concerns the shared keys of neighbors. If this shared keys are not corrupted from the start and the peers behave as the protocol states then the keys will not be leaked during a protocol run. If we assume that $key(X, Y) \not\sqsubseteq t_0$, where $X, Y \in Peers$, then at the initial state of the run there is no danger of corruption. This will help us to prove some other security properties for MUTE.

Theorem 2. Given a run of MUTE and $A_0, B_0 \in Peers(G)$, if $key(A_0, B_0) \not\sqsubseteq t_0$ then at each stage w in the run $key(A_0, B_0) \not\sqsubseteq t_w$

Proof. Suppose there is a run of MUTE in which $key(A_0, B_0)$ appears on a message sent over the network. This means, since $key(A_0, B_0) \not\sqsubseteq t_0$, that there is a stage $w > 0$ in the run st.

$$key(A_0, B_0) \not\sqsubseteq t_{w-1} \text{ and } key(A_0, B_0) \sqsubseteq t_w$$

Where $e_w \in Ev(\text{MUTE})$ (Definition 16) and by the token game of nets with persistent conditions, is st.

$$key(A_0, B_0) \sqsubseteq e_w^o$$

As can easily be checked by using the events defined in 3.4, the shape of every *Init* or *Interm* or *Resp* event

$$e \in Init : Ev(p_{Init}) \cup Interm : Ev(p_{Interm}) \cup Resp : Ev(p_{Resp})$$

is st.

$$key(A_0, B_0) \not\sqsubseteq e^o$$

The event e_w can therefore only be a spy event. If $e_w \in Spy : Ev(p_{Spy})$, however by control precedence and the token game, there must be an earlier stage u in the run, $u < w$ st. $key(A_0, B_0) \sqsubseteq t_u$ which is a contradiction. □

3.5.2.2 Secrecy property for the request

The following theorem concerns the secrecy property for the request. It states that the keyword asked by the initiator and broadcasted through the network will never be visible for a Spy outside the P2P group.

Theorem 3. Given a run of MUTE and $A_0 \in Peers(G)$ and $kw_0 \in Keywords(A_0)$, if for all peers A and $B\, key(A, B) \not\sqsubseteq t_0$, where $B \in ngb(A)$ and the run contains *Init* event a_1 labelled with action

$$act(a_1) = Init : (A_0) : i_0 : B_0 : out\, new(n_0)(\{n_0, kw_0\}_{key(A_0,B_0)}, A_0, B_0)$$

where i_0 is a session index and B_0 is an index which belongs to the set $ngb(A_0)$, n_0 is a name and kw_0 is a keyword, then at every stage w in the run $kw_0 \notin t_w$

Proof. We state a stronger property:

$$Q(p, s, t) \Leftrightarrow \sigma(kw_0, t) \subseteq \{(\{n_0, kw_0\}_{key(A_0,B_0)}, A_0, B_0)\}$$

If we can show that at every stage w in the run $Q(p_w, s_w, t_w)$ holds, then clearly $kw_0 \notin t_w$ for every stage w in the run. Suppose the contrary. By freshness clearly $Q(\text{MUTE}, s_0, t_0)$. By well-foundedness, let v be the first stage in the run st. $\neg Q(p_v, s_v, t_v)$. From the freshness principle it follows that

$$a_1 \longrightarrow e_v$$

Where $e_v \in Ev(\text{MUTE})$ (Definition 16) and from the token game $(\{n_0, kw_0\}_{key(A_0,B_0)}, A_0, B_0) \in \sigma(kw_0, t_{v-1})$ (Because messages are persistent in the net). From the token game of nets with persistent conditions we have

$$\sigma(kw_0, e_v^o - e_{v-1}^o) \not\subseteq \{(\{n_0, kw_0\}_{key(A_0,B_0)}, A_0, B_0)\} \tag{3.3}$$

Clearly e_v can only be an output event since $e_v^o - e_{v-1}^o = \emptyset$ for all input events e. Examining the output events of $Ev(\text{MUTE})$ we conclude that $e_v \notin Ev(\text{MUTE})$ reaching a contradiction.

In the following lines we will explore each output event in the protocol in order to verify that the event e_v is different to all of them.

Initiator *output events.*

$$act(e_v) = Init : (A) : j : B : out\, new(n)(\{n, kw\}_{key(A,B)}, A, B)$$

where $A \in Peers(G)$ and so $A \in s_0$ and $kw \in Keywords(A)$ and so $kw \in s_0$, where n is a name, j is a session index and B is an index which belongs to the set $ngb(A)$. Property 3.3 and the definition of message surroundings imply that $\exists\, \psi \sqsubseteq (\{n, kw\}_{key(A,B)}, A, B) \,.\, kw_0 \sqsubseteq \psi$. Then $kw_0 \sqsubseteq (\{n, kw\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and $A, B \in s_0$, freshness implies that $kw_0 \neq A$ and $kw_0 \neq B$. Since $\{n, kw\}_{key(A,B)}$ is a cyphertext, $kw_0 \sqsubseteq \{n, kw\}_{key(A,B)}$. If $kw_0 = kw$ then one reaches a contradiction to property 3.3 because from the output principle if follows that $e_v^o - e_{v-1}^o = \{\{n_0, kw_0\}_{key(A_0,B_0)}, A_0, B_0\}$. Since $kw_0 \in s_0$ freshness implies that $n \neq kw_0$. Therefore e_v cannot be an $Init$ event with the above action.

Intermediator *output events.*

$$\begin{gathered} act(e_v) = Interm : (A) : j : B : \\ out\,(\{M\}_{key(A,B)}, A, B) \end{gathered}$$

Case 1: $(M = (n, kw))$

$$act(e_v) = Interm : (A) : j : B : out\,(\{n, kw\}_{key(A,B)}, A, B)$$

where $A \in Peers(G)$ and so $A \in s_0$ and $kw \in Keywords$ and so $kw \in s_0$, where n is a name, j is a session index and B is an index which belongs to the set $ngb(A) - \{Y\}$, where $Y \in ngb(A)$ and it is the sender/forwarder of the message. Property 3.3 and the definition of message surroundings imply that $\exists\, \psi \sqsubseteq (\{n, kw\}_{key(A,B)}, A, B) \,.\, kw_0 \sqsubseteq \psi$. Then $kw_0 \sqsubseteq (\{n, kw\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$ and freshness implies that $kw_0 \neq A$ and $kw_0 \neq B$, and since $\{n, kw\}_{key(A,B)}$ is a cyphertext, $kw_0 \sqsubseteq \{n, kw\}_{key(A,B)}$. If $kw_0 = kw$ then a contradiction to property 3.3 is reached, because from the output principle if follows that $e^o_v - e^o_{v-1} = \{\{n_0, kw_0\}_{key(A,B)}, A, B\}$. Then, from the definition of message surroundings and Property 3.3 $kw_0 = n$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Interm : (A) : j : Y : in\,(\{kw_0, kw\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{kw_0, kw\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

where $kw_0 \neq n_0$ and so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ which is a contradiction since $u < v$

Case 2: $(M = (n, res, m))$

$$\begin{array}{c} act(e_v) = Interm : (A) : j : B : \\ out\,(\{n, res, m\}_{key(A,B)}, A, B) \end{array}$$

where $A \in Peers(G)$ and so $A \in s_0$ and $res \in Headers$ and so $res \in s_0$, where n, m are names, j is a session index and B is an index which belongs to the set $ngb(A) - \{Y\}$, where $Y \in ngb(A)$ and it is the sender/forwarder of the message. Property 3.3 and the definition of message surroundings imply that $\exists\,\psi \sqsubseteq (\{n, res, m\}_{key(A,B)}, A, B)\,.\,kw_0 \sqsubseteq \psi$. Then $kw_0 \sqsubseteq (\{n, res, m\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$ and freshness implies that $kw_0 \neq A$ and $kw_0 \neq B$, and since $\{n, res, m\}_{key(A,B)}$ is a cyphertext, $kw_0 \sqsubseteq \{n, res, m\}_{key(A,B)}$, and from the freshness property $kw_0 \neq res$, so if property 3.3 holds, then $kw_0 = n$ or $kw_0 = m$ and either $n \neq n_0$ or $m \neq m_0$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Interm : (A) : j : Y : in\,(\{n, res, m\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{n, res, m\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

and $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ since $(\{kw_0, res, m\}_{key(Y,A)}, Y, A) \in \sigma(kw_0, t_{u-1})$ or $(\{n, res, kw_0\}_{key(Y,A)}, Y, A) \in \sigma(kw_0, t_{u-1})$, and then $\sigma(kw_0, t_{u-1}) \not\sqsubseteq (\{n_0, kw_0\}_{key(A_0,B_0)}, A_0, B_0)$ A contradiction follows because $u < v$.

Responder *output events.*

$$act(e_v) : Resp : (A) : j : B :$$
$$out\,new(m)(\{n, res, m\}_{key(A,B)}, A, B)$$

where $A \in Peers(G)$ and so $A \in s_0$ and $res \in Headers(A)$ and so $res \in s_0$, where n, m are names, j is a session index and B is an index which belongs to the set $ngb(A)$. Property 3.3 and the definition of message surroundings imply that $\exists\, \psi \sqsubseteq (\{n, res, m\}_{key(A,B)}, A, B)\,.\, kw_0 \sqsubseteq \psi$. Then $kw_0 \sqsubseteq (\{n, res, m\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$ and freshness implies that $kw_0 \neq A$ and $kw_0 \neq B$, and since $\{n, res, m\}_{key(A,B)}$ is a cyphertext, $kw_0 \sqsubseteq \{n, res, m\}_{key(A,B)}$, and from the freshness property it follows that $m \neq kw_0$ and $res \neq kw_0$, therefore since property 3.3 holds and by definition of message surroundings $kw_0 = n$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Resp : (A) : j : in(\{kw_0, kw\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{kw_0, kw\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

Where $kw_0 \neq n_0$ and so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ which is a contradiction since $u < v$

Spy *output events.* An assumption of the theorem is that the shared keys are not leaked, meaning that for all peers A and $B\, key(A, B) \not\sqsubseteq t_0$. At every stage w in the run $key(A, B) \not\sqsubseteq t_w$ (Theorem 2). Since this there is no possible way for a spy to reach kw_0, e_v is not a spy event. $\square$

3.5.2.3 Secrecy property for the answer

The next theorem states that the message sent as an answer by the responder will never appear as a cleartext during a run of the MUTE protocol, and in this way nobody outside the peer to peer boundaries will understand it.

Theorem 4. Given a run of MUTE and $A_0 \in Peers(G)$ and $res_0 \in Headers(B_0)$, if for all peers A and $B\, key(A, B) \not\sqsubseteq t_0$, where $B \in ngb(A)$ and if the run contains a $Resp$ event b_2 labelled with action

$$act(b_2) = Resp : (A_0) : i_0 : B_0 :$$
$$out\,new(m_0)(\{n_0, res_0, m_0\}_{key(A_0,B_0)}, A_0, B_0)$$

where i_0 is a session index, B_0 is an index which belongs to the set $ngb(A_0)$, n_0, m_0 are names and $res_0 \in Headers(B_0)$ and then at every stage $w\, res_0 \notin t_w$

Proof. We show a stronger property such as this:

$$Q(p, s, t) \Leftrightarrow \sigma(res_0, t) \subseteq \{(\{n_0, res_0, m_0\}_{key(A,B)}, A, B)\}$$

If we can show that at every stage w in the run $Q(p_w, s_w, t_w)$ Then clearly $res_0 \notin t_w$ for every stage w in the run.Suppose the contrary. Suppose that at some stage in the run property Q does not hold, by freshness clearly $Q(\text{MUTE}, s_0, t_0)$. Let v by well-foundedness, be the first stage in the run st. $\neg Q(p_v, s_v, t_v)$. From the freshness principle it follows that

$$b_2 \longrightarrow e_v$$

Where $e_v \in Ev(\text{MUTE})$ (Definition 16) and from the token game $(\{n_0, res_0, m_0\}_{key(A_0,B_0)}, A_0, B_0) \in \sigma(res_0, t_{v-1})$ (messages on the network are persistent). From the token game of nets with persistent conditions the event e_v is st.

$$\sigma(res_0, e_v^o - e_{v-1}^o) \nsubseteq \{(\{n_0, res_0, m_0\}_{key(A_0,B_0)}, A_0, B_0)\} \tag{3.4}$$

Clearly e_v can only be an output event since $e_v^o - e_{v-1}^o = \emptyset$ for all input events e. We examine the possible output events of $Ev(\text{MUTE})$ and conclude that $e_v \notin Ev(\text{MUTE})$, reaching a contradiction.

In the following lines we will explore each output event in the protocol in order to verify that the event e_v is different to all of them.

Initiator *output events.*

$$act(e_v) = Init : (A) : j : B : out\, new(n)(\{n, kw\}_{key(A,B)}, A, B)$$

where $A \in Peers(G)$ and so $A \in s_0$ and $kw \in Keywords(A)$ and so $kw \in s_0$, where n is a name, j is a session index and B is an index which belongs to the set $ngb(A)$. Property 3.4 and the definition of message surroundings imply that $\exists \psi \sqsubseteq (\{n, kw\}_{key(A,B)}, A, B) \,.\, res_0 \sqsubseteq \psi$. Then $res_0 \sqsubseteq (\{n, kw\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$ and freshness implies that $res_0 \neq A$ and $res_0 \neq B$, and since $\{n, kw\}_{key(A,B)}$ is a cyphertext, $res_0 \sqsubseteq \{n, kw\}_{key(A,B)}$, and from the freshness principle it follows that $n \neq res_0$ and $res_0 \neq kw$ because $kw \in s_0$ and $kw \in Keywords$ and $res_0 \in Files$ and $Files \neq Keywords$, therefore e_v can't be a $Init$ output event with the above action.

Intermediator *output events.*

$$act(e_v) = Interm : (A) : j : B : out(\{M\}_{key(A,B)}, A, B)$$

Case 1: $(M = (n, kw))$

$$act(e_v) = Interm : (A) : j : B : \\ out\,(\{n, kw\}_{key(A,B)}, A, B)$$

where $A \in Peers$ and so $A \in s_0$ and $kw \in Keywords$ and where n is a name,j is a session index and B is an index which belongs to the set $ngb(A) - \{Y\}$ where $Y \in ngb(A)$ an it is the sender/forwarder of the message. Property 3.4 and the definition of message surroundings imply that $\exists\,\psi \sqsubseteq (\{n, kw\}_{key(A,B)}, A, B)\,.\,res_0 \sqsubseteq \psi$. Then $res_0 \sqsubseteq (\{n, kw\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$ and freshness implies that $res_0 \neq A$ and $res_0 \neq B$, and since $\{n, kw\}_{key(A,B)}$ is a cyphertext, $res_0 \sqsubseteq \{n, kw\}_{key(A,B)}$ Since $kw \in s_0$ the freshness definition implies that $res_0 \neq kw$, so $res_0 = n$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Interm : (A) : j : Y : in(\{res_0, kw\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{res_0, kw\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

where $res_0 \neq n_0$ and so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1}, res_0)$, which is a contradiction since $u < v$.

Case 2: $(M = (n, res, m))$

$$act(e_v) = Interm : (A) : j : B : \\ out\,(\{n, res, m\}_{key(A,B)}, A, B)$$

where $A \in Peers(G)$ and so $A \in s_0$ and $res \in Headers$ and so $res \in s_0$, where n, m are names, j is a session index and B is an index which belongs to the set $ngb(A) - \{Y\}$, where $Y \in ngb(A)$ and it is the sender/forwarder of the message. Property 3.4 and the definition of message surroundings implies that $\exists\,\psi \sqsubseteq (\{n, res, m\}_{key(A,B)}, A, B)\,.\,res_0 \sqsubseteq \psi$. Then $res_0 \sqsubseteq (\{n, res, m\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$ and freshness implies that $res_0 \neq A$ and $res_0 \neq B$, and since $\{n, res, m\}_{key(A,B)}$ is a cyphertext, if property 3.4 holds, then $res_0 = n$, or $res_0 = res$ or $res_0 = m$ and either $n \neq n_0$ or $res \neq res_0$ or $m \neq m_0$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

And

$$act(e_u) = Interm : (A) : j : Y : in\,(\{n, res, m\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{n, res, m\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

and $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ since either $n \neq n_0$ or $res \neq res_0$ or $m \neq m_0$. A contradiction follows because $u < v$.

Responder *output events.*

$$act(e_v) : Resp : (A) : j : B : out\, new(m)(\{n, res, m\}_{key(A,B)}, A, B)$$

where $A \in Peers(G)$ and so $A \in s_0$ and $res \in Headers(A)$ and so $res \in s_0$, where n, m are names, j is a session index and B is an index which belongs to the set $ngb(A)$. Property 3.4 and the definition of message surroundings implies that $\exists\, \psi \in (\{n, res, m\}_{key(A,B)}, A, B) \,.\, res_0 \sqsubseteq \psi$. Then $res_0 \sqsubseteq (\{n, res, m\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$ and freshness implies that $res_0 \neq A$ and $res_0 \neq B$, and since $\{n, res, m\}_{key(A,B)}$ is a cyphertext, $res_0 \sqsubseteq \{n, res, m\}_{key(A,B)}$ and the freshness property follows that $res_0 \neq m$, if $res_0 = res$ we reach a contradiction to property 3.4 because from the output principle it follows that $e_v^o - e_{v-1}^o = \{\{n_0, res_0, m_0\}_{key(A,B)}, A, B\}$. Then $res_0 = n$ By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Resp : (A) : j : in(\{res_0, kw\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{res_0, kw\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

Where $res_0 \neq n_0$ so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1}, res_0)$, which is a contradiction since $u < v$.

Spy *output events.* An assumption of the theorem is that the shared keys are not leaked, meaning that for all peers A and B $key(A, B) \not\sqsubseteq t_0$. At every stage w in the run $key(A, B) \not\sqsubseteq t_w$ (Theorem 2). Since this there is no possible way for a spy to reach kw_0, e_v is not a spy event. □

3.6 Insider Attacks, and ModMUTE

The secrecy proofs used to verify the mute protocol so far suffice to bridge the gap between formal models and ad-hoc protocols, mostly devoted to functionality instead of correctness.

However the execution environment for the protocol used in the proofs is restricted to cases where a spy is located outside the network, like a sniffer searching for packages over the internet. In a more realistic scenario, a spy can get inside the network emulating a trustful peer, exposing the network to an attack called *The man in the middle* [Hut01], acquiring or modifying the information about the request and the transfer using the direct connections between naive users. In the following section, we include a new component in the protocol, and broaden our environment including an insider spy in order to explore how the secrecy is accomplished with attacks inside the network.

3.6.1 A new component in MUTE

One of the most important changes in the protocol is the inclusion of a **File Controller Table** which aims to guarantee the consistency of the search in the protocol. Basically the file controller will hold the information about a file, its public key and a set of associated keywords, in such a way that whenever an agent requests a keyword it will immediately have the public keys related to the files which may have a relation with that particular keyword. This table is constructed by means of the union of several local tables associated to each peer in the network, which hold information about the files, their keywords and their associated public and private keys. This component is crucial in order to ensure the secrecy properties intended because, for each file in the table, a public key and a private key are generated by the owner based on unmodifiable attributes, such as a hash of the file, recording both of them in its local table, and just sharing the public one with the file controller table. In this way the inclusion of this new element in the protocol avoids attacks such as the *man in the middle*, since the only capable of understanding a request will be the one having the secret key and so the file.

File	Public Key	Keywords
file://u2/unchained melody.mp3	QGiBEOTNNcRBADYS8x/kl NNTTXTyTMa+fD4Inherin 4zvNnTR3SLebUF0447vzK ...	unchained melody , U2, The best of 1980 - 1990, Rock, etc

Figure 3.6: File Controller entry structure example

We assume that the file controller is a general entity possibly stored in a supernode, which is based on partial information present in every local table belonging to each agent in the protocol.

3.6.1.1 Presuming Confidentiality for the request keyword

We assume that any peer inside the network having a keyword kw, could immediately have the $pub(file)$, when kw belongs to $file$. So, in this fashion, that peer could send a message requesting a file, with nobody capable of understanding the message unless it had the proper file for the request. As can be seen, for an intruder to get a message or to become a real threat for any normal peer inside the network, it must have a vast amount of files to have the chance of understanding their messages. And even if the intruder did have the file, and so, its correspondant $priv(file)$, it would be very difficult for it to decrypt the message with the right key, due to its massive amount of files. This assumption turns to be true if we suppose that a normal peer has for example n files in its own store, while an intruder according to its needs, should have an obvious greater amount of files. So, while a simple peer needs a polynomial time to decrypt a message, an intruder would take an exponential time to do the work, something relatively big, that will ensure the impossibility of understanding the message, ensuring at the same time the confidentiality of it. Based on this assumption, we will state that the only one who can decrypt the message is the one who has the file, in this way we presume its good intentions.

3.6.1.2 Confidentiality for the reply

The file sent as an answer is kept as a secret because, when the recipient gets the search keyword, it also receives a public key to encrypt the answer, and the initiator is the only one with the corresponding private key to decrypt the answer.

3.6.2 Dolev-Yao Model

We recall some definitions established in 3.2.

Example: Let us consider a P2P network G with nodes A, B, X, Y, Z. Suppose that A is the initiator of the protocol. A requests a particular file it wishes to download. For this purpose it sends the request to the network by broadcasting it to its neighbors. This request includes the public key of the file $pub(file)$ associated to the search keyword $kw \in Keywords$, and the new public key $pub(s)$ associated to the initiator for encrypting the answer, which will be sent back. This message will be forwarded until it reaches a peer which has the correct file. In this case B st. $\exists f \in Files(B)$, $kw \in Keys(f)$ and kw is related to $pub(f)$. Then, B sends the answer res, st. $res \in Headers(B)$ (where res is the header of f), encrypted with the public key sent in the request by the initiator, by means of a broadcast through a series of forward steps until the target is reached, in this case sender A.

3.6.3 Specification on SPL

In this section we model an abstraction of the MUTE protocol among with the modifications related to the security for the answer and the request, stated in the proposal. We will only use a core of the protocol, just the phases involved with the transmission of the request, the answer message and the keys.

$$\begin{array}{ll}
A \longrightarrow X: & (\{N, \{pub(s)\}_{pub(file)}\}_{key(A,X)}, A, X) \text{ where } X \in ngb(A) \\
X \longrightarrow Y: & (\{N, \{pub(s)\}_{pub(file)}\}_{key(X,Y}, X, Y) \text{ where } Y \in ngb(X) \\
\ldots & \\
Y \longrightarrow B: & (\{N, \{pub(s)\}_{pub(file)}\}_{key(Y,B}, Y, B) \text{ where } B \in ngb(Y) \\
B \longrightarrow X: & (\{N, \{RES\}_{pub(s)}, M\}_{key(B,X)}, B, X) \text{ where } X \in ngb(B) \\
X \longrightarrow Y: & (\{N, \{RES\}_{pub(s)}, M\}_{key(X,Y)}, X, Y) \text{ where } Y \in ngb(X) \\
\ldots & \\
Y \longrightarrow A: & (\{N, \{RES\}_{pub(s)}, M\}_{key(Y,A)}, Y, A) \text{ where } A \in ngb(Y)
\end{array}$$

Figure 3.7: Dolev-Yao Model of the Modified MUTE protocol

$$\begin{array}{lll}
\mathbf{Init}(A) & \equiv & (\|_{B \in ngb(A)} out\, new(n,s)\, (\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)). \\
 & & (\|_{Y \in ngb(A)}\ in\, (\{n, \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A)) \\
\mathbf{Interm}(A) & \equiv & !(\|_{Y \in ngb(A)}\ in\, (\{M\}_{key(Y,A)}, Y, A).\, \|_{B \in ngb(A)-Y}\ out\, (\{M\}_{key(A,B)}, A, B)) \\
\mathbf{Resp}(A) & \equiv & (\|_{Y \in ngb(A)\,,\, file \in files(A)}\ in\, (\{X, \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A)\,. \\
 & & \|_{B \in ngb(A)}\ out\, new(m)\, (\{X, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)) \\
\mathbf{Node}(A) & \equiv & Init(A) \,\|\, Interm(A) \,\|\, Resp(A) \\
\mathbf{ModifiedSecureMute} & \equiv & \|_{A \in Peers(G)}\ Node(A)
\end{array}$$

Figure 3.8: Modified MUTE specification on SPL

For the modeling and verification process we recall the same definitions stated in 3.2.

We will state that the secrecy properties for the requests and answers do hold in the modified MUTE protocol for a spy inside the network.

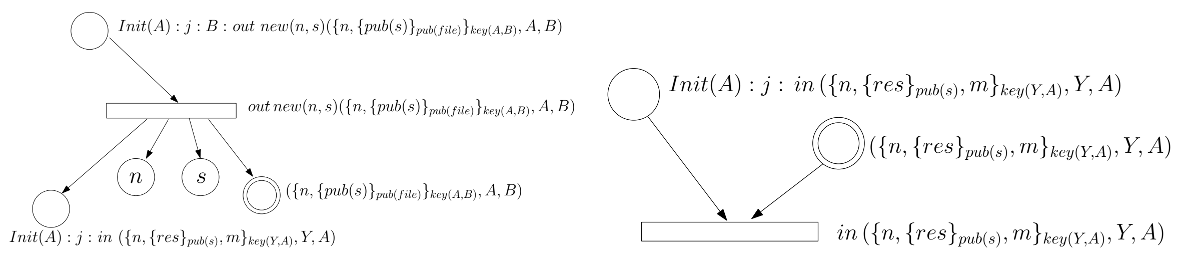
3.6.4 Events

Definition 18 (Events in ModMUTE)**.** The event e_w is in the set:
$Ev(Mod\text{MUTE}) \equiv Init : Ev(p_{Init}) \cup Interm : Ev(p_{Interm}) \cup Resp : Ev(p_{Resp}) \cup Spy : Ev(p_{Spy})$

Where the events of $Init, Interm$ and $Resp$ are defined in figures 3.9, 3.10 and 3.11.

3.6.4.1 Initiator Events

The initiator events in the modified MUTE protocol resembles the events presented for the initial version in section 3.4.1, where two kinds of actions are available: The former output action $out\, new(n,s)\, (\{n, \{pub(s)\}_{pub(file)}\}_{Key(A,B)}, A, B)$ which generates new names n, s and a request message $(\{n, \{pub(s)\}_{pub(file)}\}_{Key(A,B)}, A, B)$ directly to the store. The latter action is the input of the answer, receiving a messsage $(\{n, \{res\}_{pub(s)}, m\}_{Key(A,B)}, A, B)$ via an action $in\, (\{n, \{res\}_{pub(s)},\, m\}_{Key(A,B)}, A, B)$



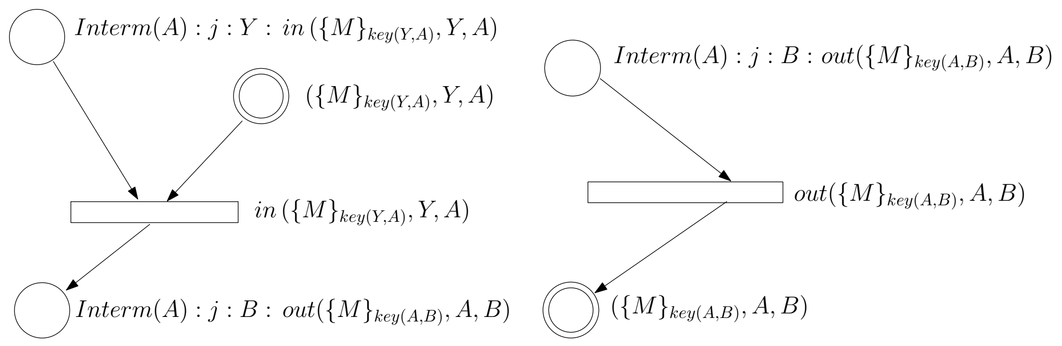
$Init(A) : j : B : out\ new(n,s)(\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$
$out\ new(n,s)(\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$
n
s
$(\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$
$Init(A) : j : in\ (\{n, \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A)$
$Init(A) : j :\ in\ (\{n, \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A)$
$(\{n, \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A)$
$in\ (\{n, \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A)$

(a) Initiator Input (b) Initiator Output

Figure 3.9: ModMUTE Initiator Events

3.6.4.2 Intermediator Events

Each agent acting as an intermediator has to forward the received messages. The figure 3.10(a) illustrates the event in which the intermediator receives the message $(\{M\}_{Key(Y,A)}, Y, A)$ via an input action $in\,(\{M\}_{Key(Y,A)}, Y, A)$. The composition of a second subprocess (figure 3.10(b)) completes the intermeditator behavior, forwarding received messages M to one of the neighbors by means of an output $out\,(\{M\}_{Key(A,B)}, A, B)$.



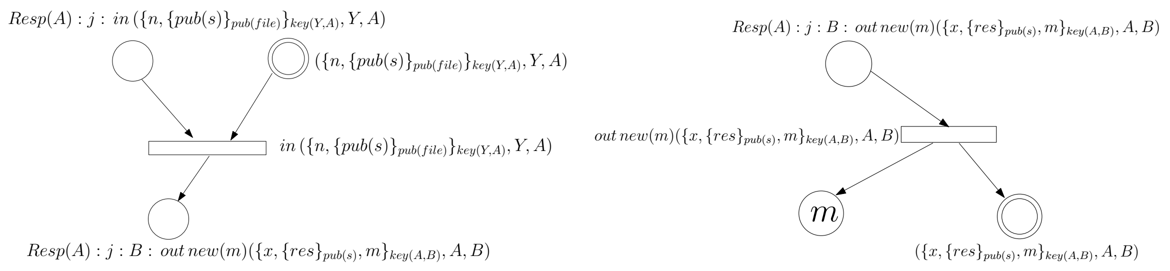
$Interm(A) : j : Y :\ in\ (\{M\}_{key(Y,A)}, Y, A)$
$(\{M\}_{key(Y,A)}, Y, A)$
$in\ (\{M\}_{key(Y,A)}, Y, A)$
$Interm(A) : j : B :\ out(\{M\}_{key(A,B)}, A, B)$
$Interm(A) : j : B :\ out(\{M\}_{key(A,B)}, A, B)$
$out(\{M\}_{key(A,B)}, A, B)$
$(\{M\}_{key(A,B)}, A, B)$

(a) Intermediator Input (b) Intermediator Output

Figure 3.10: ModMUTE Intermediator Events

3.6.4.3 Responder Events

The structure of the events for the responder in ModMUTE clearly resembles the responder events used in the original version, presented in section 3.4.3. The principal difference lies in the message structure used by the agent. The input process awaits for a message request $(\{n, \{pub(s)\}_{pub(file)}\}_{Key(Y,A)}, Y, A)$ via an input $in\,(\{n, \{pub(s)\}_{pub(file)}\}_{Key(Y,A)}, Y, A)$, and the following response output a message $(\{n, \{res\}_{pub(s)}, m\}_{Key(A,B)}, A, B)$ with a new name m.


$Resp(A) : j : in\,(\{n, \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A)$
$(\{n, \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A)$
$in\,(\{n, \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A)$
$Resp(A) : j : B : out\,new(m)(\{x, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$
$Resp(A) : j : B : out\,new(m)(\{x, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$
$out\,new(m)(\{x, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$
m
$(\{x, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$

(a) Responder Input (b) Responder Output

Figure 3.11: ModMUTE Responder Events

3.6.5 Definition of the Spy

We use the definition of a powerful spy used in 3.5.1 to model the ways of intrusion and attack that an agent can do.

$$ModifiedMUTE \equiv ModifiedSecureMUTE \| !Spy$$

3.6.6 Assumptions

Assumption 1 (Presumed Innocence): Since the only way for decrypting a message request is having the file related to the keyword. We assume that the one with the file will be a friend and will actually lend it to the requester.

Assumption 2 (Work without an end): The only way for decrypting a request is having the file. So supposing that a normal peer has n files in its own store, an intruder should have a greater amount of files, so it could have a higher possibility for decrypting the message. We assume an exponential amount of files which will give the malicious entity a higher chance of having a file, but will give it an exponential and impossible amount of work.

3.6.7 Secrecy Proofs in the Modified MUTE

To verify a security property such as secrecy behind an insider spy in P2P protocols like the modified MUTE, we must follow the same general steps used in 3.5, but with some subtle modifications that will enable to prove a much stronger property than the one verified for the MUTE protocol. In this case, the only present difference is that, since we want to guarantee a solid property such as secrecy behind an intruder which can masquerade as a trusted peer, we must also ensure in our premise, that private keys which are used to encrypt specific classified parts of the message, are never leaked to intruders inside the network.

3.6.7.1 Secrecy Properties for Shared Keys

The first secrecy theorem for the Modified MUTE protocol, regards the shared keys of neighbors. If this shared keys are not corrupted from the start and the peers behave as the protocol states, then the keys will not be leaked during a protocol run. If we assume that $\mathit{key}(X,Y) \not\sqsubseteq t_0$, where $X, Y \in \mathit{Peers}$, then at the initial state of the run, there is no danger of corruption. Later on this will help us to prove some other security properties for our protocol.

Theorem 5. Given a run of the Modified MUTE protocol and $A_0, B_0 \in \mathit{Peers}(G)$, if $\mathit{key}(A_0,B_0) \notin t_0$ then at each stage w in the run $\mathit{key}(A_0,B_0) \notin t_w$

Proof. Suppose there is a run of the Modified MUTE in which $\mathit{key}(A_0,B_0)$ appears on a message sent over the network. This means, since $\mathit{key}(A_0,B_0) \not\sqsubseteq t_0$, that there is a stage $w > 0$ in the run st.

$$\mathit{key}(A_0,B_0) \not\sqsubseteq t_{w-1} \text{ and } \mathit{key}(A_0,B_0) \sqsubseteq t_w$$

Where $e_w \in \mathit{Ev}(\mathit{Mod}\text{MUTE})$ (definition 18) and by the token game of nets with persistent conditions, is st.

$$\mathit{key}(A_0,B_0) \sqsubseteq e_w^o$$

As can easily be checked in 3.6.4, the shape of every Init or Interm or Resp event

$$e \in \mathit{Init} : \mathit{Ev}(p_{\mathit{init}}) \cup \mathit{Interm} : \mathit{Ev}(p_{\mathit{Interm}}) \cup \mathit{Resp} : \mathit{Ev}(p_{\mathit{Resp}})$$

is st.

$$\mathit{key}(A_0,B_0) \not\sqsubseteq e^o$$

The event e_w can therefore only be a Spy event. If $e_w \in \mathit{Spy} : \mathit{Ev}(p_{\mathit{Spy}})$, however by control precedence and the token game, we would find an early stage u in the run, $u < w$ st. $\mathit{key}(A_0,B_0) \sqsubseteq t_u$ and therefore we would reach a contradiction. ☐

3.6.7.2 Secrecy Property for Private Keys of the files

The second theorem for the Modified MUTE protocol is related to the private keys of the files, denoted as the decryption keys for file headers. If this private keys are not corrupted from the start of the protocol, then, they will not be leaked during a protocol run. If we assume that $\mathit{priv}(f) \not\sqsubseteq t_0$ where $f \in \mathit{files}$, then at the initial state of the run, there is no danger of corruption. This, among other theorems, will be really useful for proving more security properties in the Modified MUTE protocol.

Theorem 6. Given a run of the Modified MUTE, and $f_0 \in \mathit{files}$, if $\mathit{priv}(f_0) \not\sqsubseteq t_0$ then at each stage w in the run $\mathit{priv}(f_0) \not\sqsubseteq t_w$

Proof. Suppose there is a run of the Modified MUTE in which $\mathit{priv}(f_0)$ appears on a message sent over the network. This means, since $\mathit{priv}(f_0) \not\sqsubseteq t_0$, that there is a stage $w > 0$ in the run st.

$$\mathit{priv}(f_0) \not\sqsubseteq t_{w-1} \text{ and } \mathit{priv}(f_0) \sqsubseteq t_w$$

Where $e_w \in Ev(Mod\text{MUTE})$ (definition 18) and by the token game of nets with persistent conditions, is st.

$$\mathit{priv}(f_0) \sqsubseteq e_w^o$$

As can easily be checked, the shape of every *Init* or *Interm* or *Resp* event

$$e \in \mathit{Init} : Ev(p_{\mathit{Init}}) \cup \mathit{Interm} : Ev(p_{\mathit{Interm}}) \cup \mathit{Resp} : Ev(p_{\mathit{Resp}})$$

st. $\mathit{priv}(f_0) \not\sqsubseteq e^o$

The event e_w can therefore only be a Spy event. If $e_w \in \mathit{Spy} : Ev(p_{\mathit{Spy}})$, however by control precedence and the token game we would find an early stage u in the run, $u < w$ st. $\mathit{priv}(f_0) \sqsubseteq t_u$ and therefore we reach a contradiction. □

3.6.7.3 Secrecy Property for Private Keys of the names generated by the initiator

The third theorem for the Modified MUTE protocol regards the private keys of the names generated by the initiator. If these private keys are not corrupted from the start, and the nodes in the network behave as the protocol states, then these keys will not be leaked during a protocol run. If we assume that $\mathit{priv}(s) \not\sqsubseteq t_0$ where s is a name generated by the initiator, then at the initial state of the run there is no danger of corruption. This theorem will help us to prove some other security properties within the protocol.

Theorem 7. Given a run of the Modified MUTE and s_0 is a name generated by the initiator, if $\mathit{priv}(s_0) \not\sqsubseteq t_0$ then at each stage w in the run, $\mathit{priv}(s_0) \not\sqsubseteq t_w$

Proof. Suppose there is a run of the Modified MUTE in which $\mathit{priv}(s_0)$ appears on a message sent over the network. This means, since $\mathit{priv}(s_0) \not\sqsubseteq t_0$, there is a stage $w > 0$ in the run st.

$$\mathit{priv}(s_0) \not\sqsubseteq t_{w-1} \text{ and } \mathit{priv}(s_0) \sqsubseteq t_w$$

Where $e_w \in Ev(Mod\text{MUTE})$ (definition 18) and by the token game of nets with persistent conditions, is st.

$$priv(s_0) \sqsubseteq e^o_w$$

As can easily be checked in 3.6.4, the shape of every $Init$ or $Interm$ or $Resp$ event.

$$e \in Init : Ev(p_{Init}) \cup Interm : Ev(p_{Interm}) \cup Resp : Ev(p_{Resp})$$

is st. $priv(s_0) \not\sqsubseteq e^o$

The event e_w can therefore only be a Spy event, if $e_w \in Spy : Ev(p_{Spy})$, however by control precedence and the token game, we would find an early stage u in the run, $u < w$ st. $priv(s_0) \sqsubseteq t_u$ and therefore a contradiction is reached. □

3.6.7.4 Secrecy Property for the Request

In this case there is no searching keyword to keep as a secret. There is the need of maintaining secret the public key which will encrypt the answer file and which will be sent back to the initiator. This should be kept as a secret to guarantee that the one who will use it to encrypt the answer, is the real owner of the file.

The following theorem states that the request, broadcasted by the protocol initiator, will never be visible by any peer inside the network, unless it has the the real answer to that request.

Theorem 8. Given a run of the Modified MUTE and $A_0 \in Peers(G)$ and $f_0 \in files$, if for all peers A and $B\, key(A, B) \not\sqsubseteq t_0$, where $B \in ngb(A)$, and $priv(f_0) \not\sqsubseteq t_0$, assuming *the presumed innocence* and *the work without an end* assumptions, the run contains an $Init$ event a_1 labeled with action

$$act(a_1) = Init : (A_0) : i_0 : B_0 : out\, new(n_0, s_0)\, (\{n_0, \{pub(s_0)\}_{pub(f_0)}\}_{key(A_0,B_0)}, A_0, B_0)$$

Where i_0 is a session index, B_0 is an index which belongs to the set $ngb(A_0)$, n_0 and s_0 are names, f_0 a file and $pub(s_0)$ an encrypting public key, then at every stage w in the run $pub(s_0) \notin t_w$

Proof. We state a stronger property such as this:

$$Q(p, s, t) \Leftrightarrow \sigma(pub(s_0), t) \subseteq \{(\{n_0, \{pub(s_0)\}_{pub(f_0)}\}_{key(A,B)}, A_0, B_0)\}$$

If we can show that in every stage $Q(p_w, s_w, t_w)$. Then clearly $pub(s_0) \notin t_w$ for every stage w in the run. Suppose that at some stage in the run the property does not hold. Let v, by well-foundedness, be the first stage in the run st. $\neg Q(p_v, s_v, t_v)$. From the freshness principle it follows that

$$a_1 \longrightarrow e_v$$

and from the token game $\{(\{n_0, \{pub(s_0)\}_{pub(f_0)}\}_{key(A_0,B_0)}, A_0, B_0)\} \in \sigma(pub(s_0), t_{v-1})$ (Because messages are persistent in the net). Where $e_v \in Ev(Mod\text{MUTE})$ (definition 18) and from the token game of nets with persistent conditions is st.

$$\sigma(pub(s_0), e^o_v - e^o_{v-1}) \not\subseteq \{(\{n_0, \{pub(s_0)\}_{pub(f_0)}\}_{key(A_0,B_0)}, A_0, B_0)\} \tag{3.5}$$

Clearly e_v can only be an output event since $e^o_v - e^o_{v-1} = \emptyset$ for all input events e. Examining the output events of $Ev(Mod\text{MUTE})$ we conclude that $e_v \notin Ev(Mod\text{MUTE})$ reaching a contradiction.

In the following lines we will explore each output event in the protocol in order to verify that the event e_v is different to all of them.

Initiator *output events.*

$$act(e_v) = Init : A : j : B : out\, new(n, s)\, (\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$$

Where $A \in Peers(G)$ so $A \in s_0$ and $file \in files$ so $file \in s_0$. Where n and s are names, $pub(s)$ is a public key associated to the name s, j is a session index and B is an index which belongs to the set $ngb(A)$, and $B \in Peers(G)$ so $B \in s_0$. Property 3.5 and the definition of message surroundings imply that $pub(s_0) \sqsubseteq (\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$, freshness implies that $pub(s_0) \neq A$ and $pub(s_0) \neq B$. Since $\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}$ is a cyphertext, $pub(s_0) \sqsubseteq \{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}$. From the freshness principle it follows that $pub(s_0) \neq n$ and since $\{pub(s_0)\}_{pub(file)}$ is a cyphertext, $pub(s_0) \sqsubseteq \{pub(s)\}_{pub(file)}$. If $pub(s_0) = pub(s)$ then one reaches a contradiction to property 3.5 because from the output principle it follows that $e^o_v - e^o_{v-1} = (\{n_0, \{pub(s_0)\}_{pub(f_0)}\}_{key(A_0,B_0)}, A_0, B_0)$. Therefore e_v cannot be an $Init$ event with the above action.

Intermediator *output events.*

$$act(e_v) = Interm : (A) : j : B : out\, (\{M\}_{key(A,B)}, A, B)$$

Case 1: $(M = (n, \{pub(s)\}_{pub(file)}))$

$$act(e_v) = Interm : (A) : j : B : out\, (\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$$

Where $A \in Peers(G)$ so $A \in s_0$ and $file \in files$ so $file \in s_0$. Where n and s are names, $pub(s)$ is a public key associated to the name s, j is a session index and B is an index which belongs to the set $ngb(A) - Y$, and $B \in Peers(G)$ so $B \in s_0$. Property 3.5 and the definition of message surroundings imply that $pub(s_0) \sqsubseteq (\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$, freshness implies that $pub(s_0) \neq A$ and $pub(s_0) \neq B$. Since $\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}$ is a cyphertext, $pub(s_0) \sqsubseteq \{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}$. Since $\{pub(s)\}_{pub(file)}$ is a cyphertext and property 3.5 must hold, we first say that $pub(s_0) = n$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

And

$$act(e_u) = Interm : (A) : j : Y : in\,(\{pub(s_0), \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A)$$

By the token game

$$\{pub(s_0), \{pub(s)\}_{pub(file)}\}_{key(Y,A)} \in t_{u-1}$$

Where $pub(s_0) \neq n_0$ then $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$. A contradiction follows because $u < v$.

Since property 3.5 has not been fulfilled and $\{pub(s)\}_{pub(file)}$ is a cyphertext, $pub(s_0) \sqsubseteq \{pub(s)\}_{pub(file)}$. If $pub(s_0) = pub(s)$ then a contradiction to property 3.5 is achieved because from the output principle it follows that $e_v^o - e_{v-1}^o = (\{n_0, \{pub(s_0)\}_{pub(f_0)}\}_{key(A_0,B_0)}, A_0, B_0)$. Therefore e_v cannot be an $Interm$ event.

Case 2: $(M = (n, \{res\}_{pub(s)}, m))$

$$act(e_v) = Interm : (A) : j : B : out\,(\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$$

Where $A \in Peers(G)$ so $A \in s_0$ and $res \in Headers$ so $res \in s_0$. Where n, m and s are names, $pub(s)$ is a public key associated to the name s, j is a session index and B is an index which belongs to the set $ngb(A) - Y$, and $B \in Peers(G)$ and so $B \in s_O$. Property 3.5 and the definition of message surroundings imply that $pub(s_0) \sqsubseteq (\{n\{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$. Since $A \in s_0$ and $B \in s_0$, freshness follows that $pub(s_0) \neq A$ and $pub(s_0) \neq B$ and since $\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}$ is a cyphertext, $pub(s_0) \sqsubseteq \{n, \{res\}_{pub(s)}, m\}_{key(A,B)}$. Since $\{res\}_{pub(s)}$ is a cyphertext and since property 3.5 must hold, and by the definition of message surroundings, we first say that $pub(s_0) = n$ or $pub(s_0) = m$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

And

$$act(e_u) = Interm : (A) : j : Y : in\,(\{n, \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A)$$

By the token game

$$\{n, \{res\}_{pub(s)}, m\}_{key(Y,A)} \in t_{u-1}$$

and $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ since $(\{pub(s_0), \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A) \in \sigma(pub(s_0), t_{u-1})$ or $(\{n, \{res\}_{pub(s)}, pub(s_0)\}_{key(Y,A)}, Y, A) \in \sigma(pub(s_0), t_{u-1})$, and then $\sigma(pub(s_0), tu-1) \not\subseteq \{(\{n_0, \{pub(s_0)\}_{pub(f_0)}\}_{key(A,B)}, A_0, B_0)\}$. A contradiction follows because $u < v$.
Since property 3.5 has not been fulfilled and $\{res\}_{pub(s)}$ is a cyphertext, $pub(s_0) \sqsubseteq \{res\}_{pub(s)}$, then $pub(s_0) = res$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

And

$$act(e_u) = Interm : (A) : j : in : Y : (\{n, \{pub(s_0)\}_{pub(s)}, m\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{n, \{pub(s_0)\}_{pub(s)}, m\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

Where $pub(s_0) \neq res_0$ and so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$. A contradiction follows because $u < v$.

Responder *output events.*

$$act(e_v) = Resp : (A) : j : B : out\,new(m)\,(\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$$

Where $A \in Peers(G)$ so $A \in s_0$ and $res \in files$ so $res \in s_0$. Where n, m and s are names, $pub(s)$, is a public key associated to the name s, j is a session index and B is an index which belongs to the set $ngb(A)$, and $B \in Peers(G)$ so $B \in s_0$. Property 3.5 and the definition of message surroundings imply that $pub(s_0) \sqsubseteq (\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$, freshness implies that $pub(s_0) \neq A$ and $pub(s_0) \neq B$, and since $\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}$ is a cyphertext, $pub(s_0) \sqsubseteq \{n, \{res\}_{pub(s)}, m\}_{key(A,B)}$. By the property of freshness $m \neq pub(s_0)$ and since $\{res\}_{pub(s)}$ is a cyphertext and property 3.5 must hold, and by the definition of message surroundings, we first state that $pub(s_0) = n$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

And

$$act(e_u) = Resp : (A) : j : in\,(\{pub(s_0), \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{pub(s_0), \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

Where $pub(s_0) \neq n_0$ and so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$. A contradiction follows because $u < v$. Since property 3.5 has not been fulfilled and $\{res\}_{pub(s)}$ is a cyphertext, $pub(s_0) \sqsubseteq \{res\}_{pub(s)}$. Since $res \in s_0\, pub(s) \neq res$ Therefore e_v cannot be an $Resp$ event with the above action.

Spy *output events.* An assumption of the theorem is that the shared keys and the private key of the file are not leaked, meaning that for all peers A and $B\, key(A, B) \not\sqsubseteq t_0$ and $priv(f_0)$ $\not\sqsubseteq t_0$. At every stage w in the run $key(A, B), priv(f_0) \not\sqsubseteq t_w$ (Theorems 5, 6). Since this, there is no possible way for a spy to reach $pub(s_0)$, e_v is not a spy event. □

3.6.7.5 Secrecy Property for the Answer

This theorem establishes that the answer, sent by the responder peer, will be kept as a secret for every peer different from the initiator.

Theorem 9. Given a run of the Modified MUTE and $A_0 \in Peers(G)$ and $res_0 \in files(A_0)$, if for all peers A and $B\, key(A, B) \not\sqsubseteq t_0$, where $B \in ngb(A)$, $priv(s) \not\sqsubseteq t_0$ and the run contains a $Resp$ event a_2 labeled with action

$$act(a_2) = Resp : (A_0) : i_0 : out\, new(m_0)\, (\{n_0, \{res_0\}_{pub(s_0)}, m_0\}_{key(A_0,B_0)}, A_0, B_0)$$

Where i_0 is a session index, B_0 is an index which belongs to the set $ngb(A)$, $n_0\, m_0$, are names and $res_0 \in files(B_0)$ and then at every stage $w\, res_0 \not\sqsubseteq t_w$.

Proof. We show a stronger property such as this

$$Q(p, s, t) \Leftrightarrow \sigma(res_0, t) \subseteq \{(\{n_0, \{res_0\}_{pub(s_0)}, m_0\}_{key(A_0,B_0)}, A_0, B_0)\}$$

If we can show that at every stage w in the run $Q(p_w, s_w, t_w)$ then clearly $res_0 \not\sqsubseteq t_w$ for every stage in the run, property Q does not hold, by freshness clearly $(Mod\text{MUTE}, s, t)$, Let v by well-foundedness, be the first stage in the run st. $\neg Q(p_v, s_v, t_v)$. From the freshness principle it follows that

$$a_2 \longrightarrow e_v$$

and from the token game $(\{n_0, \{res_0\}_{pub(s_0)}, m_0\}_{key(A_0,B_0)}, A_0, B_0) \in \sigma(res_0, t_{v-1})$ (Because messages are persistent in the net). Where $e_v \in Ev(Mod\text{MUTE})$ (definition 18) and from the token game of nets with persistent conditions the event e_v is st.

$$\sigma(res_0, e_v^o - e_{v-1}^o) \not\subseteq \{(\{n_0, \{res_0\}_{pub(s_0)}, m_0\}_{key(A_0,B_0)}, A_0, B_0)\} \quad (3.6)$$

Clearly e_v can only be an output event since $e_v^o - e_{v-1}^o = \emptyset$ for all input events e. We examine the possible output events of $Ev(Mod\text{MUTE})$ and conclude that $e_v \not\in Ev(Mod\text{MUTE})$, reaching a contradiction.

In the following lines we will explore each output event in the protocol in order to verify that the event e_v is different to all of them.

Initiator *output events.*

$$act(e_v) = Init : j : B : out\,new(n, s)\,(\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$$

Where $A \in Peers(G)$ so $A \in s_0$ and $file \in files$. Where n and s are names, $pub(s)$ is a public key associated to the name s, j is a session index and B is an index which belongs to the set $ngb(A)$, and $B \in Peers(G)$ and so $B \in s_0$. Property 3.6 and the definition of message surroundings imply that $res_0 \sqsubseteq (\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$, freshness implies that $res_0 \neq A$ and $res_0 \neq B$, and since $\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}$ is a cyphertext, $res_0 \sqsubseteq \{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}$. From the freshness principle it follows that $res_0 \neq n$ and since $\{pub(s_0)\}_{pub(file)}$ is a cyphertext, $res_0 \sqsubseteq \{pub(s)\}_{pub(file)}$. By the property of freshness $res_0 \neq pub(s)$. Then e_v cannot be an $Init$ event with the above action.

Intermediator *output events.*

$$act(e_v) = Interm : (A) : j : B : out\,(\{M\}_{key(A,B)}, A, B)$$

Case 1: $(M = (n, \{pub(s)\}_{pub(file)}))$

$$act(e_v) = Interm : (A) : j : B : out\,(\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$$

Where $A \in Peers(G)$ so $A \in s_0$, where n and s are names, $pub(s)$ is a public key associated to the name s, j is a session index and B is an index which belongs to the set $ngb(A) - Y$, and $B \in Peers(G)$ and so $B \in s_O$. By property 3.6 and the the definition of message surroundings it follows that $res_0 \sqsubseteq (\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$, freshness implies that $res_0 \neq A$ and $res_0 \neq B$, and since $\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}$ is a cyphertext, $res_0 \sqsubseteq \{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}$. Since $\{pub(s)\}_{pub(file)}$ is a cyphertext and property 3.6 must hold, we first say that $n = res_0$. By control precedence there exists an event e_u in the run st.

$$\begin{tikzcd} e_u \arrow[r] & e_v \end{tikzcd}$$

And

$$act(e_u) = Interm : (A) : j : Y : in\,(\{res_0, \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{res_0, \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

Where $res_0 \neq n_0$ and so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$. A contradiction follows because $u < v$. Since property 3.6 has not been fulfilled and $\{pub(s)\}_{pub(file)}$ is a cyphertext, $res_0 \sqsubseteq \{pub(s)\}_{pub(file)}$. Then $res_0 = pub(s)$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

And

$$act(e_u) = Interm : (A) : j : Y : in(\{n, \{res_0\}_{pub(file)}\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{n, \{res_0\}_{pub(file)}\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

Where $res_0 \neq pub(s)$ and so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$. A contradiction follows because $u < v$.
Case 2: $(M = (n, \{res\}_{pub(s)}, m))$

$$act(e_v) = Interm : (A) : j : B : out(\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$$

Where $A \in Peers(G)$ so $A \in s_0$, where n, m and s are names, $pub(s)$ is a public key associated to the name s, j is a session index and B is an index which belongs to the set $ngb(A) - Y$, and $B \in Peers(G)$ and so $B \in s_O$. Property 3.6 and the definition of message surroundings imply that $res_0 \sqsubseteq (\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$. Since $A \in s_0$ and $B \in s_0$, freshness follows that $res_0 \neq A$ and $res_0 \neq B$ and since $\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}$ is a cyphertext, $res_0 \sqsubseteq \{n, \{res\}_{pub(s)}, m\}_{key(A,B)}$. Since $\{res\}_{pub(s)}$ is a cyphertext and property 3.6 must hold, by the definition of message surroundings, we first state that $res_0 = n$ or $res_0 = m$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Interm : (A) : j : Y : in(\{n, \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A)$$

By the token game

$$(\{n, \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A) \in t_{u-1}$$

and $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ since $(\{res_0, \{res\}_{pub(s)}, m\}_{key(Y,A)}, Y, A) \in \sigma(res_0, t_{u-1})$ or $(\{n, \{res\}_{pub(s)}, res_0\}_{key(Y,A)}, Y, A) \in \sigma(res_0, t_{u-1})$, and then $\sigma(res_0, tu-1) \not\subseteq \{(\{n_0, \{res_0\}_{pub(s_0)}, m_0\}_{key(A_0,B_0)}, A_0, B_0)\}$. A contradiction follows because $u < v$.
Since property 3.6 has not been fulfilled and $\{res\}_{pub(s)}$ is a cyphertext, $res_0 \sqsubseteq \{res\}_{pub(s)}$. If $res_0 = res$ then one reaches a contradiction to property 3.6 because from the output principle it follows that $e_v^o - e_{v-1}^o = (\{n_0, \{res_0\}_{pub(s_0)}, m_0\}_{key(A_0,B_0)}, A_0, B_0)$. Therefore e_v cannot be an *Interm* event with the above action.

Responder *output events.*

$$act(e_v) = Resp : (A) : j : B : out\,(\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$$

Where $A \in Peers(G)$ so $A \in s_0$, where n, m and s are names, $pub(s)$ is a public key associated to the name s, j is a session index and B is an index which belongs to the set $ngb(A)$, and $B \in Peers(G)$ and so $B \in s_O$. By property 3.6 and the the definition of message surroundings it follows that $res_0 \sqsubseteq (\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}, A, B)$. Since $A, B \in Peers(G)$ and then $A, B \in s_0$ and freshness implies that $res_0 \neq A$ and $res_0 \neq B$, and since $\{n, \{res\}_{pub(s)}, m\}_{key(A,B)}$ is a cyphertext, $res_0 \sqsubseteq \{n, \{res\}_{pub(s)}, m\}_{key(A,B)}$. By the freshness property $res_0 \neq m$. Since $\{res\}_{pub(s)}$ is a cyphertext and property 3.6 must hold, we first say that $n = res_0$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Resp : (A) : j : in\,(\{res_0, \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A)$$

By the token game, $(\{res_0, \{pub(s)\}_{pub(file)}\}_{key(Y,A)}, Y, A) \in t_{u-1}$, where $res_0 \neq n_0$ and so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$. A contradiction follows because $u < v$.

Since property 3.6 has not been fulfilled and $\{res\}_{pub(s)}$ is a cyphertext, $res_0 \sqsubseteq \{res\}_{pub(s)}$. If $res_0 = res$ then a contradiction to property 3.6 is reached, because from the output principle it follows that $e_v^o - e_{v-1}^o = (\{n_0, \{res_0\}_{pub(s_0)}, m_0\}_{key(A_0,B_0)}, A_0, B_0)$. Therefore e_v cannot be an $Resp$ event with the above action.

Spy *output events.* An assumption of the theorem is that the shared keys and the new private key generated by the initiator are not leaked, meaning that for all peers A and B $key(A,B) \not\sqsubseteq t_0$ and $priv(s_0) \not\sqsubseteq t_0$. At every stage w in the run $key(A_0,B_0), priv(s_0) \not\sqsubseteq t_w$ (Theorems 2, 7). Since this, there is no possible way for a spy to reach $pub(s_0)$, e_v is not a spy event. □

3.7 Discussion

Along this chapter we have shown two significant contributions relevant to the work on security. The first one relates to the generality of SPL. To the authors knowledge, process calculi for security protocols are intensively used in the analysis and verification of security properties like authentication, secrecy, non-malleability and non-repudiation. In specific, SPL process calculus was used in the verification of middle-size authentication examples (see [CCM02, CW01, Cra03] for further information). However, an industrial-size protocol, including a high amount of message-exchanges and a great number of agents involved, was never modelled. We bear witness of the flexibility and generality of SPL reasoning techniques by using them in a large size protocol.

The second contribution is related to the modification of the MUTE protocol, in order to tackle attacks, directly from the core of the network. This includes several design decisions that can be considered intrusive in the main protocol idea, such as the inclusion of a file controller. However, there are facts that increase the security of the system instead of diminishing it: Not publishing the file contents and the association of them, to widely known tuples of public keys/ keywords allows only the owners of the file to detect the requests for an specific file. This approach has other advantages as well, the reduction of the message length in the protocol increases the network performance, and multiple sources of the files can be discovered if the keys are generated based on a seed that uses integrity checks of each file (ie. a hash function).

3.8 Summary

Along this chapter we consider our efforts to analyze the security properties of the MUTE protocol. To formally model the MUTE protocol for the first time, we have abstracted security aspects directly from the source code, considering only those concerned to security, such as key management and ciphering of public channels to model link encryptions. This abstraction does not consider every phase involved in the protocol, but intends to compile the most crucial interactions where leakage of information is critical.

With the formal specification of the protocol, we have used SPL operational semantics and its general proof principles to state the secrecy property by dividing it into three specific phases: The distribution of shared keys, the communication of the request and the final response. The basic idea underlying these proof techniques was to state hypothetical events not fullfilling the stated properties, and by means of the operational semantics show that those events can never be reached generating a contradiction.

Although MUTE was only intended to ensure secrecy for outsider agents, we went further and include two basic modifications of the protocol in order to fullfill the secrecy property in environments where agents inside the network can become untrustful. This basic changes includes the creation of a new entity that maintains the information of the files without publishing their contents deliberately, and a completely improved protocol that adds a middle phase, enabling a secure search using encrypted messages with keys associated to files, instead of the usual file contents. This modified protocol was also verified in the same way as it was done with the authentic MUTE protocol.

4 Exploring Integrity and Secrecy Issues over a P2P collaborative System

Collaborative P2P applications aim to allow application-level collaboration between users. The inherently ad-hoc nature of P2P technology makes it a good fit for user-level collaborative applications. These applications range from instant messaging and chat, to on-line games, to shared applications [Ese02, BS04, GK03, BMWZ05, Rip01] that can be used in business, educational, and home environments. Unfortunately, a number of technical challenges remain to be solved before pure P2P collaborative implementations become viable, such as location discovery, fault tolerance, network constraints and security [MKL$^+$02]. Concerning to the security of the system, P2P systems are used to share private information between peers over open networks, involving properties like secrecy, anonymity and non-traceability which have been studied in the literature in order to overcome such risks [MKL$^+$02].

In chapter 3 we showed how SPL can be a suitable framework for the analysis of security aspects of P2P protocols. In this chapter we explore how can SPL reasoning techniques can serve as well for the analysis of a collaborative P2P system. We use a cutting-edge system as a valid case of study to achieve this affirmation. This system is intended to resolve the problem of automatic reconfiguration of applications in a fully distributed system without compromising the identities of the agents involved in the protocol, neither their own secrets.

We follow a two-fold approach for tackling the problem of dynamic reconfiguration of applications in P2P systems. Firstly, we extend the basic syntactic structure of SPL with some notions of concurrency relevant to security to formalize an SPL model for the Friends Troubleshooting Network (FTN) protocol [HWB05]. Secondly, we propose a new protocol that maintains the main functionality of the FTN protocol, in a model much concise and less complex than the proposed by Wang et al. In order to do so, we heavily use the idea of a layered encryption protocol [GRS99].

The chapter is structured as follows. In section 4.1 we explain the problem of dynamic reconfiguration of P2P-based applications, taking the FTN network architecture as base. A definition of the essential properties to be ensured in this kind of systems is given in section 4.2. In section 4.3 we extend the basic syntax of SPL by means of a set of encodings, to enable a formal model for the FTN protocol. Then in section 4.4 we give a formalization for a new and more concise protocol with the same functionality as FTN. The DR protocol is verified using the basic proof structure inherent to SPL.

4.1 Dynamic Reconfiguration Systems

The problem of dynamic reconfiguration of systems is inherent to a wide variety of problems such power consumption networks [DGOR04], agent networks [PR99], and P2P systems [WHY$^+$04]. The problem addresses the inconveniences present where a distributed and highly dynamic system need to modify the states of each agent without loss of information. In this section, we explain in deeper detail this problem based in an specific problem of P2P systems: The reconfiguration of applications in P2P systems.

4.1.1 FTN protocol

The Friends Troubleshooting Network (FTN) is a protocol that explores the advantages of P2P approach in automatic reconfiguration of applications [HWB05]. Placing in context, the protocol operates in an open environment where the correct behavior of each agent depends on a configuration table, where stored entries are comformed by a key attribute and a privacy sensitive record value.

Basically the protocol sends the request of a misconfigured application and the suspicious entries that possibly origin the problem to a group of trusted agents (friends). They contribute to solve the problem revising its own records in search for suspects according the request and updating the vector of suspects modifying the probability for each suspect, as well including their own suspects. The protocol continues in the way that each friend could request for aid to his own friends, spreading the process until a fixed number of agents has collaborated in the request. Finally, each friend involved in the protocol returns backward the vector of suspects until the requester is reached, and he only has to repair the suspect entry with more probability.

There are several security aspects that we have to consider: First, relating to integrity, we must ensure that nobody can alter the contents of a given message. Second we must guarantee that nobody can trace the origin of the request. Third, that nobody can guess which entries are included or modified for some agent, and finally: that only the agents that are trusted must include information in the request.

4.1.1.1 Agent Definition

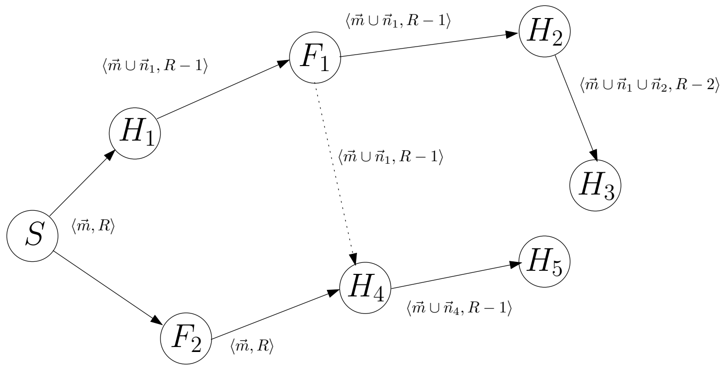
An agent in the P2P system can be either a *sick* machine, a *helper* or a *forwarder*. Each one of these roles is explained next.

Sick Machine The first step to make a request for the sick machine is to convert the privacy sensitive information (e.g., login/password information, credit card numbers, and so on) into widely known constants that preserve the semantics of the message. Then the requester must send to one of the trusted friends the request including the vector of suspicious entries mapped before, the name of the misconfigured application, a new name to identify the request, and the number of hops (network jumps) needed to end the search. Finally the sick machine awaits for confirmation of the friend. If a confirmation message is received, the protocol simply waits for the eventually response of his friend and operates consequently, subtracting from the vector the value with most probability. Otherwise he chooses another friend and repeat the process.

Friend Machine The first thing that a friend agent does after receiving the request information, is to choose whether to help or not to the requester. This is done by sending the respectively acknowledge to the requester. The next step is to decide what role the friend is going to take in the protocol: to help modifying and including information into the vector of suspicious entries, or to forward the request to another friend. If he wants to help in the request, the friend operates over the vector of suspicious entries adding its own suspects and incrementing values into previous entries based on his local reasoning. The main aspect in this process is to help the sick machine without revealing its own applications. Finally, the friend verifies if it is the last hop in the protocol, sending the message forward if there are remaining hops waiting, or backward if is the last agent in the protocol.

Forwarder The forwarder simply selects one of his own friends and passes away the request, expecting their response for a limited time. If it arrives, he sends it backwards, otherwise he must cancel forward requests and send the trace to the previous agent in the protocol.

An example of the protocol is illustrated in Figure 4.1, where a sick machine S publishes his request to his friends H_1 and F_1 which are intended to participate in the request helping and forwarding the data. Each agent that helps in the request includes information to the vector of suspect entries (as seen in the output messages of H_1, H_2 and H_4). If an agent has already collaborate in a request, it stops the input request (denoted as a dotted line between F_1 and F_4). Also, the protocol ends when a fixed number of collaborative agents are involved in the protocol (in this case, the value is limited by 3) sending the response backwards, so other traces that cannot reach this level will be stuck and the friend agents can never reply to the sender their values.



$\langle \vec{m} \cup \vec{n}_1, R-1 \rangle$
H_2
$\langle \vec{m} \cup \vec{n}_1, R-1 \rangle$
F_1
$\langle \vec{m} \cup \vec{n}_1 \cup \vec{n}_2, R-2 \rangle$
H_1
$\langle \vec{m} \cup \vec{n}_1, R-1 \rangle$
H_3
S
$\langle \vec{m}, R \rangle$
H_5
H_4
$\langle \vec{m} \cup \vec{n}_4, R-1 \rangle$
F_2
$\langle \vec{m}, R \rangle$

Figure 4.1: Friends Troubleshooting Network Model

4.1.1.2 Known Attacks

This version of the protocol evidence two types of attacks which are covered broadly by Wang et al. at [HWB05]. The first of them, called *Gossip Attack*, shows that a collusion between two non-immediate agents in the protocol could infer what are the new messages posted for the agent in the middle, as shown in figure 4.2.

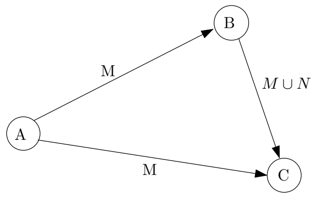


Figure 4.2: Gossip Attack: C could infer the contents added by B

Another attack that could break the secrecy of the messages in the protocol is *the polling attack*. This attack could make use of the parameter denoting the number of remaining hops needed to discover the secrets added by the last agent for the previous agent involved in the protocol.

4.1.2 Characteristics of Fixed FTN

With this considerations in mind, a new version of FTN was released [WHY+04]. The main characteristic of the protocol fix includes the concept of shared spaces: each helper that wants to contribute into the protocol, must create a cluster with his own friends, sharing the messages of the request and modifying or publishing his own request into the cluster.

The procedure for the cluster is explained as follows. First, the cluster entrance B receives the message M, then it publishes M to its friends in order to establish the cluster. When the cluster is properly established, B publishes M in the shared space and the agents in order that his friends could have access to the request. In this way the cluster members could publish the results of their own consults using M. Every agent in the system is a trusted friend so we can say that the information of the cluster is not used for his own purposes, and the number of messages included by each agent in the cluster depends of its own local computation. Finally, one of the cluster members forward the messages contained in the cluster in order to continue with the protocol. The image 4.3 illustrates the process above.

Another of the corrections included in the revision of FTN was to change the number of hops: No agent could know where is the last hop in the network. This could be made adding probability to the protocol changing R to $1 - \frac{1}{N}$ where N is the minimum number of samples needed, and the stop condition is modified so it stops when the probability $P(1 - \frac{1}{N}) \approx 0$

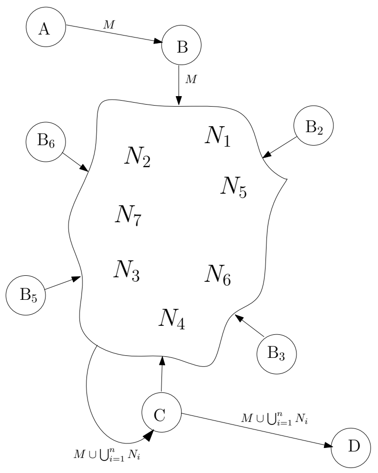


Figure 4.3: Cluster Modeling

4.2 Security properties to be Assured

In this model, we must describe the security properties in order to prove the correctness and functionality of the protocol:

- **Integrity:** This property states that contents in a message must persist all over the life cycle of the message delivery. This means that any kind of information can be added to the message, but without altering its old contents. More formally, for every message response M' in transit from peer A to B the integrity of this message is ensured if $M \sqsubseteq M'$ such that M' is the message generated just before M. In this way we ensure a monotonic message, which is always part of the next generated message. Due to the importance of the answers from the agents involved in the request, we must ensure that:
  - Every data included by a friend peer into the answer, must remain until reaching the protocol requester agent.
- **Secrecy:** Also known as *Anonymity beyond suspicion*[MKL+02]. Ensures that the real information published by an agent can never be known by other peers in the network, different from its target. Formalizing, for every message going from A to B, the information published is never showed as a cleartext, or as cyphertext which can be decrypted by other peers rather than the both mentioned before, during the delivery life cycle. In this way, we must show that:
  - The plain text m created by an initiator agent A can never be derived from other messages in the protocol.
  - The plain text x created by a friend agent B can never be derived from other messages in the protocol.

4.3 A close FTN approach with SPL

As we have explained, this protocol includes several concurrency considerations that involve security, such as the exclusive choice of roles, cluster handling, and mutable spaces in the protocol. These features are not defined in SPL, basically relying on limitations concerned to the inherent model of the persistent store. However, this class of constructions are widely provided and used in other process calculi such as π [Mil99] or Spi[AG97a]. In this section we provide a set of encodings to achieve these task, formalizing FTN network architecture as a well grounded example where this concepts remains crucial.

4.3.1 Encodings

4.3.1.1 Exclusive Non-deterministic choice

The choice between two excluding processes is not a new idea. This operator was introduced by Milner [Mil99], Abadi & Fournet [AF01] and Palamidessi & Valencia [PV01], and intends to represent the execution of a process with tasks with the same possibility of being executed. This idea differs from the parallel composition in the way that if one of them is selected, the other processes remains stuck stopping their evolution over time. However, the concept of parallel composition, new nonces, and message exchange can serve as well for achieving this task, for example, given a process R with two exclusive choices P and Q:

- A public key f is generated and distributed to P and Q in order to guarantee the freshness of the choice.
- Both processes generate a fresh public key that is sent to a common process which selects one key according to the time of arrival, responding with a fresh name encrypted with the public key received.
- The process receiving the response will be the one which will execute, while the other will remain stuck forever.

Clearly, if a third process R is involved in a sequential composition, it has to wait until one of the process is completely executed. With the considerations presented before we present the formal model of this construction in SPL:

$$\begin{aligned}(P+Q).R \equiv\ & out\,new(f,g)\{Pub(f)\}_{Pub(g)}\,.\,(out\,new(s)\,\{Pub(s)\}_{Pub(f)}\,.\,in\,\{x\}_{Pub(s)}\,.\,P.R\,\|\\ & out\,new(t)\,\{Pub(t)\}_{Pub(f)}\,.\,in\,\{x\}_{Pub(t)}\,.\,Q.R\,\|\\ & in\,\{Pub(Z)\}_{Pub(f)}\,.\,out\,new(a)\,\{a\}_{Pub(Z)}\end{aligned} \tag{4.1}$$

4.3.1.2 Indexed Exclusive non-deterministic choice

A non-deterministic choice behavior over a set of process P can be generalized from the previous encoding in the following way:

$$\begin{aligned}(\|_{+\, i\in\{1..n\}} P_i).R \equiv\, & out\, new(f,g)\{Pub(f)\}_{Pub(g)} \,.\, (\|_{i\in\{1..n\}} out\, new(s)\, \{Pub(s)\}_{Pub(f)} \,.\, in\, \{x\}_{Pub(s)} \,.\, P_i.R) \,\| \\ & in\, \{Pub(Z)\}_{Pub(f)} \,.\, out\, new(a)\, \{a\}_{Pub(Z)}\end{aligned} \tag{4.2}$$

It relies in the same concepts stated in 4.3.1.1

If a process R has to be executed strictly after an indexed non deterministic choice $\|_{+\, i\in\{1..n\}} P_i$ we adopt the same idea as in equation 4.1.

4.3.1.3 Indexed sequential composition

SPL presents an indexed parallel composition process by which represents several indexed processes working in parallel. Despite being a very important concept for concurrency, sometimes the need of ensuring that all processes will execute one after another and not at the same time arises. For example, taking a subtle modified version of the Readers and Writers mutual exclusion problem [CHP71]. In our own instance of the problem, every writer executes his task before the execution of the reader. In this particular case, the only problem arises when two or more writers want to modify the shared resource at the same time. Therefore, since every writer must execute its job having exclusive access to the critical section, we must ensure some kind of order in the set, in a way that while some agent is writing, the others just wait for their turn. A simple sequential composition between writing processes is not an adequate solution, due to the amount of processes that should be written in order to complete the whole composition, so we must make use of the new concept of indexed sequential composition.

Therefore, we will make some minimal changes to the parallel composition in such a way that we can turn it into a sequential composition.

$$\begin{aligned}\|_{seq\, i\in\{1..n\}} P_i \quad \equiv \quad & out\, new(a)\, \{a\}_{Key(P_0,P_1)} \,.\, (\|_{i\in\{1..n-1\}}\, in\, \{x\}_{Key(P_{i-1},P_i)} \,.\, P_i \,.\, out\, \{x\}_{Key(P_i,P_{i+1})}) \,. \\ & in\, \{x\}_{Key(P_{n-1},P_n)} \,.\, P_n\end{aligned} \tag{4.3}$$

Explanation In this encoding, the key factor are the shared keys between the components inside the parallel composition. These keys will work as channels by which the indexed elements will communicate in a way that each one will trigger the execution of the other.

- We have an output process outside the parallel composition which will start the execution of the indexed processes. This can be easily seen because for the first indexed process to get started, it first has to receive an acknowledge through the channel it shares with the initial process outside the parallel composition.
- In the same way, after the first process inside the parallel composition executes, it will send an acknowledge via the channel it shares with the following process and this one will have to wait until receiving it.
- The last component works outside the parallel composition. It awaits until receiving the acknowledge from the last process, to get started.

4.3.1.4 Sequential replication

In the same way as a Replication is an infinite parallel composition of processes, a sequential replication is an infinite indexed sequential composition of processes. This kind of process is needed when a processes must be executed infinitely, one after the other.

$$!_{seq}P \quad \equiv \quad out\, new(a)\, \{a\}_{Key(P_0,P_1)} \cdot \|_{i\in\{1..\infty\}}\, in\, \{x\}_{Key(P_{i-1},P_i)} \cdot P_i \cdot out\, \{x\}_{Key(P_i,P_{i+1})} \tag{4.4}$$

Explanation It relies in the same concepts stated in 4.3.1.3

4.3.2 Modeling a Cluster for FTN

Since SPL has a monotonic store, which means that messages output into the network persist forever, it turns to be really difficult to model a cluster by means of this language, requiring an space with mutable capacity. Then, modeling an abstraction of a mutable space on this calculus must be done via the encodings stated in 4.3.1. A mutable cluster in SPL can be seen as a store with several instances through its life time. Therefore, we model a store in which each time the messages of the cluster are modified, another instance is created, with a different and new lock which will identify the store, denying the access from intruders. The keys and locks to the space of messages will be managed by a the cluster initiator, updating the keys and redirecting the spaces each time the cluster is modified, assigning a turn for each of the principals involved. The cluster is a composition of two main processes, Initiator and Participants.

4.3.2.1 Initiator

This process initiates the cluster by generating its first instance. Following, it triggers the execution of the next component participating in the cluster.

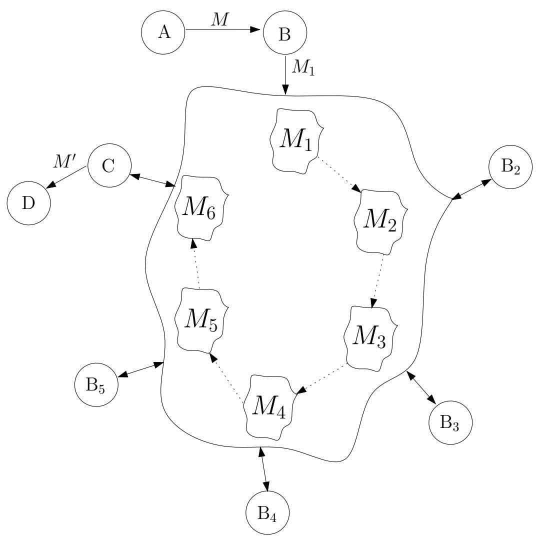


Figure 4.4: Cluster over a persistent network : The store evolves by means of linked stores

$$\begin{aligned} Initiator(A,B,M) \quad &\equiv \quad out\,new(k)\,\{k\}_{Pub(A)}\,.\,fun(A,Pub(k),M)\,. \\ &\quad\;\; in\,\{M'\}_{Pub(k)}\,.\,out\,new(a)\,\{M'\}_{Pub(a)}\,.\,out\,\{Priv(a)\}_{Key(A,B)} \end{aligned} \tag{4.5}$$

Where A represents the cluster initiator which works as the keeper or manager of the cluster, M the message by which the first information of the cluster is generated and B the next component participating in the cluster. $fun(A, Pub(k), M)$ will represent the function by which the agent A generates a message M' by computing the already received message M with its own local information in a single tuple. (see section 5.3 for further details). The tuple M' generated by this function must include the contents of M. Finally the message M' is output to the space of messages encrypted with $Pub(k)$.

- Here A generates the first key ($Priv(a)$) and lock ($Pub(a)$) for the cluster. Then introduces the initial information inside the cluster, generated by means of the function stated before, and encrypts it with the lock ($Pub(a)$).
- Afterwards, agent A sends the key ($Priv(a)$) to the next participant B, so it can modify the cluster.

4.3.2.2 Participants

This process models the behavior of the rest of agents participating in the cluster. It represents the way in which each peer interacts with the cluster by collaborating or not collaborating with the cause.

$$
\begin{aligned}
Participants(A,\vec{P},n) \quad \equiv \quad & (\|_{seq\, v\in\{1..n-1\}}\,(in\{Priv(Z)\}_{Key(A,P_v)}\,.\,in\,\{M\}_{Pub(Z)}\,.\\
& (Contributor(P_v,A,M,Priv(Z))\;+Non-Contributor(P_v,A,Priv(Z)))\,.\\
& Distributor(A,P_v)))\,.\,in\{Priv(Z)\}_{Key(A,P_n)}\,.\,in\,\{M\}_{Pub(Z)}\,.\\
& (Contributor(P_n,A,M,Priv(Z))\;+\;Non-Contributor(P_n,A,Priv(Z)))
\end{aligned}
$$

where

$$
\begin{aligned}
Contributor(A,B,M,Priv(Z)) \quad & \equiv \quad out\,new(k)\,\{k\}_{Pub(A)}\,.\,fun(Pub(k),A,M)\,.\,in\{M'\}_{Pub(k)}\,.\\
& \qquad out\,new(b)\,\{M'\}_{Pub(b)}\,.\,out\,\{Priv(b),Priv(Z)\}_{Key(B,A)}\\
Non-Contributor(A,B,Priv(Z)) \quad & \equiv \quad out\,\{Priv(Z),Priv(Z)\}_{Key(B,A)}\\
Distributor(A,P_m) \quad & \equiv \quad in\,\{Priv(X),Priv(Y)\}_{Key(A,P_m)}\,.\\
& \qquad out\,\{Priv(X)\}_{Key(A,P_{m+1})}
\end{aligned}
$$

Figure 4.5: Cluster Formalization

Where A is the cluster manager, $\vec{P}$ the rest of friends participating in the cluster and n the cardinality of $\vec{P}$.

- The first agent receives key $(Priv(Z))$ and opens the store for the information inside it.
- If this peer does not want to modify any content inside the cluster, it just executes $Non-Contributor$ and sends back the same key to the server which will pass it to the next process. But, if the agent wants to modify the contents of the cluster it executes the $Contributor$ process, by which it generates a new key $(Priv(b))$ and lock $(Pub(b))$ and calls the function $fun(P,Pub(k),M)$ by which M' is generated. The agent receives M' encrypted with $Pub(k)$, decrypts it and locks it with its previously generated lock, $(Pub(b))$. Then, it sends the key $(Priv(b))$ to the server which will continue, and forward that new key to the next participant in the cluster by means of the $Distributor$ process.

Putting all together, the cluster can be formalized as:

$$
Cluster(S,\vec{P},n,init) \quad \equiv \quad Initiator(S,P_1,init)\,.\,Participants(S,\vec{P},n) \tag{4.6}
$$

4.3.3 Assumptions

In order to model the FTN protocol among these encodings, we have to include some assumptions for the reader's understanding. We focus on the modeling of anonymous communications in a well established network, so we consider a model where authentication between peers was previously done using an authentication protocol. Let $Peers(G)$ represent the whole P2P network as in 3.2, and $f(S)$ the set of friends of any agent S.

Definition 19 (FTN Messages)**.** Let $I \equiv (Rid, init)$ an initial message, where Rid is a message identifier and $init$ a tuple which will include all the information required for the initiator to request some help. The initial message I evolves through the protocol in the following way $I \rightarrow I' ... \rightarrow L$, where $I' \equiv (Rid, (init, M))$ and M is the new information added by each friend in the protocol, which decides to help the requester. An finally $L \equiv (Rid, (init, M), end)$ represents the last message sent back to the initiator via the same path where it arrived. In this last message the name *end*, known by every peer in the network, is included to identify this specific message as the one which has to be sent backwards until reaching the requester process.

4.3.4 Requester behavior

The requester or initiator A generates a message with the following structure:

$$\{Rid, init\}_{Key(A,X)} \text{ Where } X \in f(A)$$

In this way the requester sends the message of all of its friends, which will decide if the will help it or will just forward its request.

- Request Output: $out\, new(Rid, init)\, \{Rid, init\}_{Key(A,X)}$ where $X \in f(A)$. The output request will be sent to every friend of the requester, encrypted with shared key between friends, in such a way that the only one which can understand the message are the group of friends of the initiator.
- Reception of the answer: $in\, \{Rid, (init, M), end\}_{Key(Y,A)}$. The requester receives as an answer the first message received by one of its friends including the name *end*, which will mean that the data recollection have ended, and the message now includes the information required for the solution of its problem.

This behavior is condensed in figure 4.6: (Recalling the message structure presented in definition 19)

$$Init(A) \quad = \quad (\|_{i \in f(A)}\, out\, new(Rid, init)\, \{I\}_{Key(A,i)})\, .\, in\{L\}_{Key(A,i)}$$

Figure 4.6: Model of a Requester

4.3.5 Helper Agent Behavior

The first action that a friend has to resolve is to help or to forward. If the agent decide to help, it generates a cluster with a group of trusted friends in such a way that inside this store, a great amount of information can be recollected. Then, the helper selects one of the principals involved in the cluster and pass the control over the information received in the cluster, one of the friends sharing the cluster, takes out the last information remaining. This agent has two similar options to take, either it may forward this information to another friend which will decide to help or not, or it can just send back the recollected data to the originator of the cluster, in a way that it eventually the protocol initiator can be reached.

- Decision: The agent which takes out the information from the cluster, has to decide between continue helping (by forwarding the data took out from the cluster), or just sending back the information to the cluster initiator, which will redirect it until it reaches the protocol initiator. This is done in a non-deterministic way by means of the choice encoding: $HelperFwd(A) + HelperBckwd(A)$.
- Reception of the request: $in\{Rid, M\}_{Key(Y,A)}$ Here the helper receives the message capturing the information needed to proceed, with its help in the variable M.
- Cluster Help: $Cluster(S, f(A), n, (Rid, M))$. Here the helper generates a cluster by which it will recollect information to help the initiator of the process. The helper always acts as manager S which will be in charge of the cluster.
- Taking out the information from the *Cluster*: As we have seen in the *Cluster* encoding (section 4.3.2), the number n agent in this collaborating process is the last participant and so, the one with the last private key $Priv(x)$. Therefore, it is the one receiving the information encrypted with that key $in\,\{(Rid, (init, M))\}_{Pub(x)}$
- Continue helping: In the process $HelperFwd(A)$ a chosen helper takes out the information from the cluster ($f(A)_n$ the n friend of A which participates in the cluster process) and decides to forward the message to other friends, waiting for a response which it will send back to the cluster initiator.
- Sending Back: In process $HelperBckwd(A)$ the chosen helper ($f(A)_n$) takes out the data from the cluster and sends back the information to the cluster initiator which redirects it back.

With this considerations, the helper is modelled in figure 4.7

4.3.6 Forwarder Role

- Forwards the request and waits for the response in order to return it to the sender. The model of this agent is shown in Figure 4.8 (We recall the message structure presented in definition 19)

$$\begin{array}{lcl} HelperFwd(A) & = & (\|_{i \in f(f(A)_n)}\ out\,\{I'\}_{Key(f(A)_n,i)})\,.\,in\,\{L\}_{Key(f(A)_n,i)}\,. \\ & & out\,\{L\}_{Key(f(A)_n,A)}\,.\,in\,\{L\}_{Key(f(A)_n,A)}\,.\,out\,\{L\}_{Key(A,Y)} \\ & & \\ HelperBckwd(A) & = & out\,\{L\}_{Key(f(A)_n,A)}\,.\,in\,\{L\}_{Key(f(A)_n,A)}\,.\,out\,\{L\}_{Key(A,Y)} \\ & & \\ Helper(A) & = & Collaborate(A).\,(HelperFwd(A)\,+\,HelperBckwd(A)) \\ \text{Where} & & \\ Collaborate(A) & = & \|_{Y \in f(X)}\ in\,\{I'\}_{Key(Y,A)}\,.\,Cluster(A, f(A), n, I')\,.\,in\,\{I'\}_{Pub(x)} \end{array}$$

Figure 4.7: Model of a Helper

$$Fwd(A) \quad = \quad !(in\,\{I'\}_{Key(Y,A)}\,.\,(\|_{i \in f(A)}\,out\,\{I'\}_{Key(A,i)})\,.\,in\,\{L\}_{Key(i,A)}\,.\,out\,\{L\}_{Key(A,Y)})$$

Figure 4.8: Model of a Forwarder

4.3.7 The FTN Protocol

Putting all together, the instance of the protocol is modelled below:

$$\begin{array}{lcl} Node(A) & = & Init(A)\,\|\,Fwd(A)\,\|\,Helper(A) \\ & & \\ FTN & = & \|_{A \in Peers(G)}\,Node(A) \end{array}$$

Figure 4.9: Instance of FTN Protocol

Here we have the FTN protocol, where the initiator is the sick machine which wants to be helped. It sends a collaboration message to all its friends, which will either help it, or forward the request in their own behalf, to one of their friends. If the friend of the initiator or just a subsequent friend wants to help, it will call all its own friends and will organize a cluster. There, all participants will make a brainstorm and will recollect information which can be sent back to the initiator through the same path, or could be moved forward in a search for more information.

4.4 Dynamic Reconfiguration Protocol: an FTN simplified protocol

In this model, we pretend to conserve the functionality of the system and the main security properties with a model strictly close to SPL, with a much more simpler protocol. The Dynamic Reconfiguration protocol (DR), modifies the way each agent interacts with ideas inspired in multiple encryption stages, as in the Onion routing protocol [GRS99]. In this way, we abstract certain aspects of the protocol, like the use of an anonymity function in order to fulfill the requirements imposed. We will represent a P2P network using the definition 15.

The intuitive description of the protocol is presented below:

In this scheme, the initiator agent A creates a request, with a new identifier Rid, a new

$$\begin{array}{ll}
A \longrightarrow X: & \{R, Pub(k), \{M\}_{Pub(k)}\}_{key(A,X)} \text{ where } X \in f(A) \\
X \longrightarrow Y: & \{R, Pub(k), \{\{M\}_{Pub(k)}, P\}_{Pub(k)}\}_{key(X,Y)} \text{ where } Y \in f(X) \\
... & \\
Y \longrightarrow B: & \{R, Pub(k), \{M', P\}_{Pub(k)}\}_{key(Y,B)} \text{ where } B \in f(Y) \\
B \longrightarrow X: & \{N, R, Pub(k), M'\}_{key(B,X} \text{ where } X \in f(B) \\
X \longrightarrow Y: & \{N, R, Pub(k), M'\}_{key(X,Y)} \text{ where } Y \in f(X) \\
... & \\
Y \longrightarrow A: & \{N, R, Pub(k), M'\}_{key(Y,A)} \text{ where } A \in f(Y)
\end{array}$$

Figure 4.10: Dolev-Yao Model of the DR protocol

public key $Pub(k)$ and a new secret $\{M\}_{Pub(k)}$. It sends the request encrypted with a shared key $Key(A, P_1)$ to a friend agent P_1. In this way, P_1 receives the information sent by A and includes into the request his own information, ciphering it with the public key sent in the request. This process is made for each agent present in the protocol, constructing a ciphered-layer message, only possible to discover for the owner of the key (A in this case). This process continue until the last helper agent in the protocol includes its own information in the request, sending back the response message using the same path where he had received the request, ciphering the message with key $Pub(k)$. Finally, A receives the message and recurrently decrypts the message until it reaches his own genererated identifier nonce, verifying the integrity of the information if the secret is inside the response.

4.4.1 DR Formalization

Definition 20 (Layered Messages)**.** Every message ψ in the DR protocol has a shape: $\psi \in \{\{m\}_{Pub(k)}, \{\{m\}_{Pub(k)}, p\}_{Pub(k)}, \{\{\{m\}_{Pub(k)}, p\}_{Pub(k)}, p\}_{Pub(k)}, ...\}$ Where m is the variable in which the nonce identifier generated by the request should go, $Pub(k)$ is the public key generated by the initiator of the protocol and p is the variable in which the information included by each helper should remain.

Definition 21 (Submessages under any level of encryptions)**.** Let $\psi\langle x\rangle$ be a message where $x \gg \psi$. Where $x \gg \psi$ is a relation defined in the following way: $x \gg \psi$ if $x \sqsubseteq \psi \vee \exists\psi_0$ st. $\psi_0 \sqsubseteq \psi \wedge \psi_0 \neq \psi \wedge x \gg \psi_0$

Definition 22 (Encryption Seed)**.** Let $\psi\langle x\rangle[x/m]$ a message where x appears under any level of encryptions but just substituting the m variable inherent to the message shape.

Definition 23 (FTN Sets)**.** Let $Info$ represents the data owned by all peers in the network, $Info(X)$ the information belonging specifically to peer X, $f(X)$ represents the set of friends of peer X, and $Peers(G)$ the set of all peers in the network. In our model we assume that $Key(X, Y) = Key(Y, X)$

The protocol consists of an interaction between two kind of processes, $Alice(X)$ and $Bob(X)$. $Alice(X)$ is declared as an initiator agent that first creates the request identifier Rid, and

$$\begin{array}{lcl}
Alice(X) & \equiv & (\|_{i\in f(X)}\, out\, new\,(Rid,k,m)\left\{Rid, Pub(k), \{m\}_{Pub(k)}\right\}_{Key(X,i)}) \\
 & & .\, in\left\{n, Rid, Pub(k), \psi\langle\{m\}_{Pub(k)}\rangle\right\}_{Key(i,X)} \\
Bob(X) & \equiv & (\|_{Y\in f(X)}\, in\,\{res\}_{Key(Y,X)}\,.\,(Fwd(X,Y,res) \parallel Triumph(X,Y,res))) \\
Node(X) & \equiv & Alice(X) \parallel Bob(X) \\
DR* & \equiv & \|_{X\in Peers(G)}\, Node(x)
\end{array}$$

Where

$$\begin{array}{lcl}
Fwd(X,Y,res) & \equiv & (\|_{j\in f(X)}\, out\left\{Rid, Pub(k), \left\{\{m\}_{Pub(k)}, p\right\}_{Pub(k)}\right\}_{Key(X,j)}) \\
 & & .\, in\,\{n, res\}_{Key(j,X)}\,.\,out\,\{n, res\}_{Key(X,Y)} \\
Triumph(X,Y,res) & \equiv & out\, new\,(n)\left\{n, Rid, pub(k), \{\psi, p\}_{Pub(k)}\right\}_{Key(X,Y)}
\end{array}$$

And

$$res \equiv (Rid, Pub(k), \psi)$$

Figure 4.11: SPL model of DR protocol

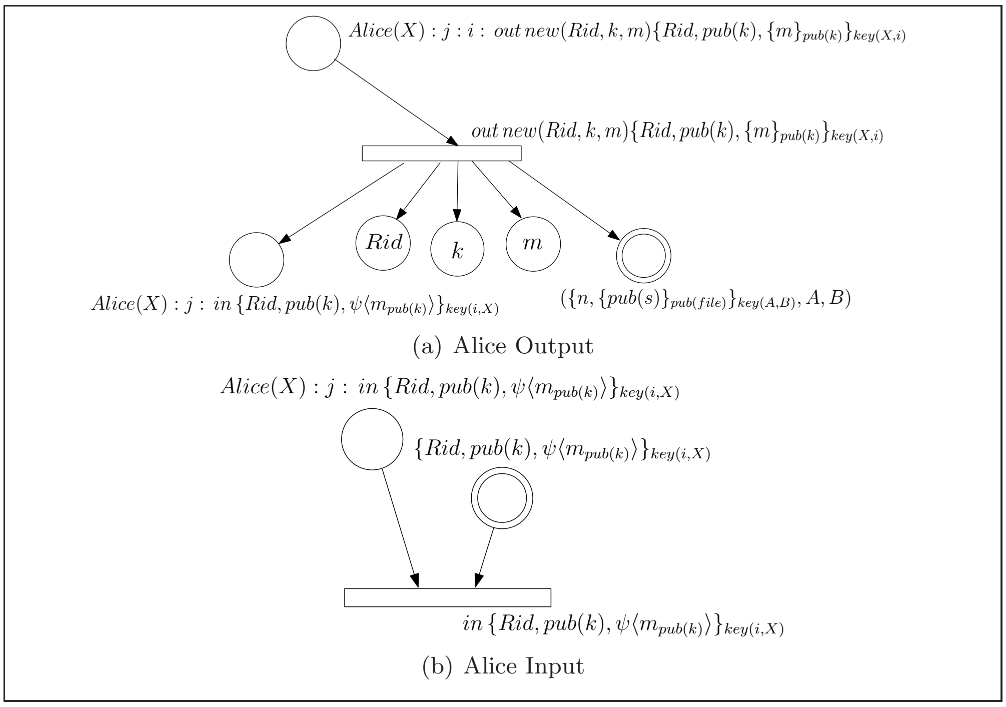
a new pair of names k, m, then sends the request message to his friends including the fresh name m encrypted with $Pub(k)$ among with Rid. Finally, the agent expects for a reception message with the responses p encrypted in a multilayer system, with all the layers ciphered using the public key of k, including the encrypted fresh name m sent previously and a new name n which identifies the message as an answer.

$Bob(X)$ denotes a friend agent that receives the request information and operates forwarding the response message with his own suspects to one of its friends, ciphering the tuple that contains the contents received previously and the new message in a new encryption layer with the public key of k. It also can send the multi-layered encryption response immediately to the initiator, among with a new name n which denotes that the message shall go back through the same path it came in. The concrete model of the DR protocol can be seen in figure 4.11

4.4.2 Events

4.4.2.1 Alice Events

Alice events represent the actions available for a general requester in the friends network. Alice is composed by two subprocesses : An output process (fig. 4.12(a)), where Alice sends a message $\{Rid, pub(k), \{m\}_{pub(k)}\}_{key(X,i)}$ requesting for help to any of her friends in $f(x)$, generating new names Rid, k, m. The second action available for Alice is the reception of an answer contained in the message $\{Rid, pub(k), \psi\langle\{m\}_{pub(k)}\rangle\}_{key(i,X)}$ via an action $in\,\{Rid, pub(k), \psi\langle\{m\}_{pub(k)}\rangle\}_{key(i,X)}$ (fig. 4.12(b)).


$Alice(X) : j : i :$ $out\,new(Rid, k, m)\{Rid, pub(k), \{m\}_{pub(k)}\}_{key(X,i)}$
$out\,new(Rid, k, m)\{Rid, pub(k), \{m\}_{pub(k)}\}_{key(X,i)}$
Rid
k
m
$Alice(X) : j :$ $in\,\{Rid, pub(k), \psi\langle m_{pub(k)}\rangle\}_{key(i,X)}$
$(\{n, \{pub(s)\}_{pub(file)}\}_{key(A,B)}, A, B)$
$Alice(X) : j :$ $in\,\{Rid, pub(k), \psi\langle m_{pub(k)}\rangle\}_{key(i,X)}$
$\{Rid, pub(k), \psi\langle m_{pub(k)}\rangle\}_{key(i,X)}$
$in\,\{Rid, pub(k), \psi\langle m_{pub(k)}\rangle\}_{key(i,X)}$

(a) Alice Output

(b) Alice Input

Figure 4.12: *Alice* Events

4.4.2.2 Bob Events

An execution of the agent *Bob* can be branched in a number of sub-processes: the initial event done is the reception of a request message $\{Rid, pub(k), \psi\}_{key(Y,X)}$ from any of the friends in $f(A)$ via an input action $in\,\{Rid, pub(k), \psi\}_{key(Y,X)}$. At this point, *Bob* can evolve in one of the sub-process of forwarding or response transmission.

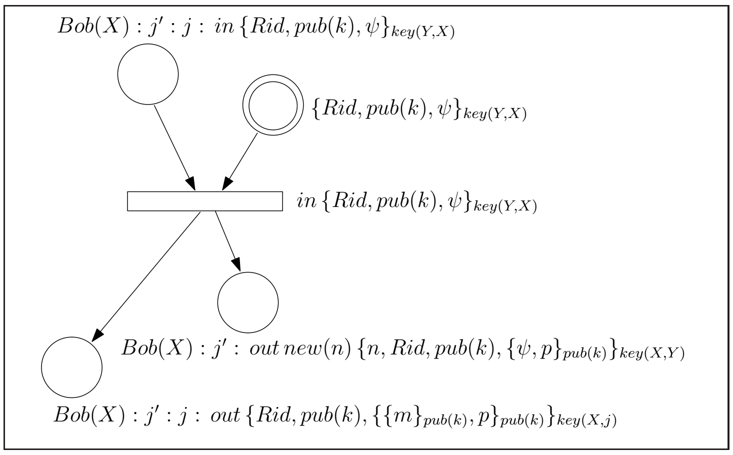

$Bob(X) : j' : j :$ $in\,\{Rid, pub(k), \psi\}_{key(Y,X)}$
$\{Rid, pub(k), \psi\}_{key(Y,X)}$
$in\,\{Rid, pub(k), \psi\}_{key(Y,X)}$
$Bob(X) : j' :$ $out\,new(n)\,\{n, Rid, pub(k), \{\psi, p\}_{pub(k)}\}_{key(X,Y)}$
$Bob(X) : j' : j :$ $out\,\{Rid, pub(k), \{\{m\}_{pub(k)}, p\}_{pub(k)}\}_{key(X,j)}$

Figure 4.13: *Bob* Initial Event

4.4.2.3 Forwarder Events

Forwarder events indicate those events in which Bob helps contributing with the request and sending the modified message to a friend for further assistance. It is basically composed by three sub-processes: The first process (fig. 4.14(a)) generates an output event with the message $\{Rid, pub(k), \{\{m\}_{pub(k)}, p\}_{pub(k)}\}_{key(X,j)}$ via an output action $out\,\{Rid, pub(k), \{\{m\}_{pub(k)}, p\}_{pub(k)}\}_{key(X,j)}$. The next action available for the forwarder generates an input event for the answer messages $\{n, Rid, pub(k), \psi\}_{key(j,X)}$, sent back towards the originator agent *Alice* (fig. 4.14(b)). Finally, the last action (fig. 4.14(c)) generates an output event with the message $\{n, Rid, pub(k), \psi\}_{key(X,Y)}$ by means of the output action $out\,\{n, Rid, pub(k), \psi\}_{key(X,Y)}$.

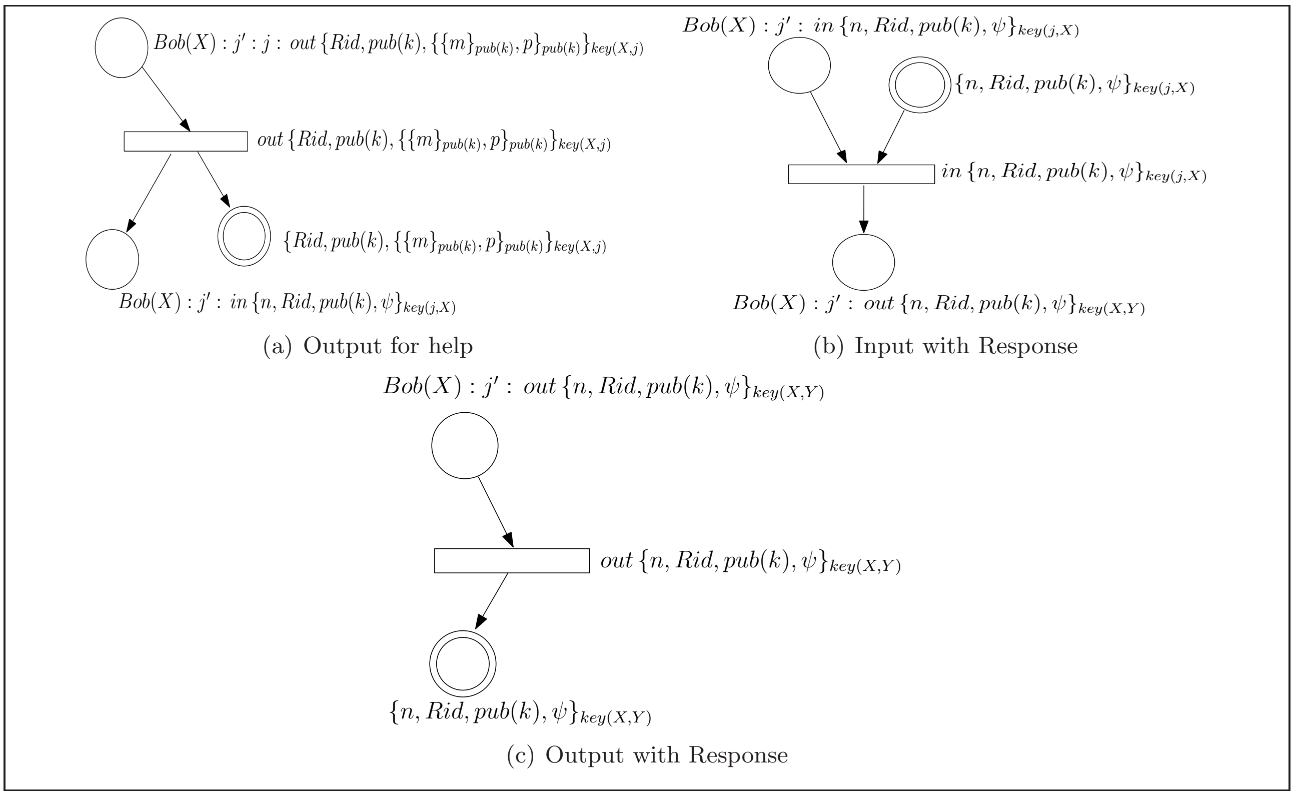

$Bob(X) : j' : j : out\,\{Rid, pub(k), \{\{m\}_{pub(k)}, p\}_{pub(k)}\}_{key(X,j)}$
$out\,\{Rid, pub(k), \{\{m\}_{pub(k)}, p\}_{pub(k)}\}_{key(X,j)}$
$\{Rid, pub(k), \{\{m\}_{pub(k)}, p\}_{pub(k)}\}_{key(X,j)}$
$Bob(X) : j' : in\,\{n, Rid, pub(k), \psi\}_{key(j,X)}$
(a) Output for help
$Bob(X) : j' : in\,\{n, Rid, pub(k), \psi\}_{key(j,X)}$
$\{n, Rid, pub(k), \psi\}_{key(j,X)}$
$in\,\{n, Rid, pub(k), \psi\}_{key(j,X)}$
$Bob(X) : j' : out\,\{n, Rid, pub(k), \psi\}_{key(X,Y)}$
(b) Input with Response
$Bob(X) : j' : out\,\{n, Rid, pub(k), \psi\}_{key(X,Y)}$
$out\,\{n, Rid, pub(k), \psi\}_{key(X,Y)}$
$\{n, Rid, pub(k), \psi\}_{key(X,Y)}$
(c) Output with Response

Figure 4.14: Forwarder Events

4.4.2.4 Triumph Event

Bob Triumph event indicates the event in which the help ends, generating a message $\{\psi, p\}_{pub(k)}\}_{key(X,Y)}$ with a new name n, with the action $out\,new(n)\,\{n, Rid, pub(k), \{\psi, p\}_{pub(k)}\}_{key(X,Y)}$, as can be seen in figure 4.15

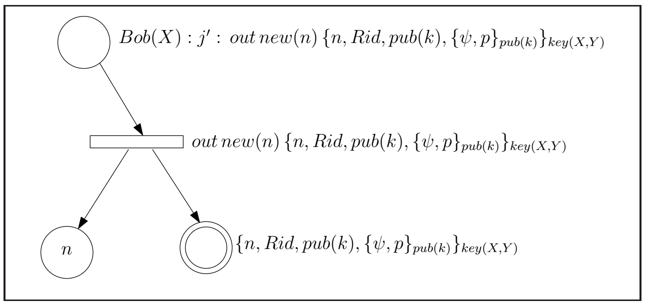

$Bob(X) : j' : \; out\, new(n)\, \{n, Rid, pub(k), \{\psi, p\}_{pub(k)}\}_{key(X,Y)}$
$out\, new(n)\, \{n, Rid, pub(k), \{\psi, p\}_{pub(k)}\}_{key(X,Y)}$
n
$\{n, Rid, pub(k), \{\psi, p\}_{pub(k)}\}_{key(X,Y)}$

Figure 4.15: *Triumph* Event

4.4.3 Definition of the Spy

We use the definition of a powerful spy used in SPL (section 3.5.1) to model the ways of intrusion and attack that an agent can do.

$$DR \equiv DR * \| !Spy$$

4.4.4 Secrecy Proofs in DR

To ensure the secrecy property for the response messages in the FTN protocol, we must follow a set of general steps.

Initially, we must verify that the private keys used for encrypting the information added by each helper, are never leaked during message transmissions. This fact is relevant in order to assure that the data added by a friend who wants to help the sick machine, could be understood only by the initiator peer.

Then, assuming that those keys are never leaked, this secrecy property can be proved in a straightforward way, by presenting a stronger property which states that every response message added by a friend, is encrypted with a private key only known by the initiator of the protocol, and since we know that messages encrypted with these keys can never be decrypted by other rather than the peer requesting for help, the secrecy property for responses is fulfilled. In order to verify this property, each output event occurring in the protocol must be verified, to ensure that there is no message responses from friends intended for the initiator, which appear in non ciphered messages.

4.4.4.1 Secrecy Property for Private Key generated by the initiator

The first theorem for the DR protocol regards the private key generated by the initiator. If this private key is not corrupted from the start, and the nodes in the network behave as the protocol states, then this key will not be leaked during a protocol run. If we assume that

$Priv(k) \not\sqsubseteq t_0$ where k is a name generated by the initiator, then at the initial state of the run there is no danger of corruption. This theorem will help us to prove some other security properties within the protocol.

Theorem 10. Given a run of the DR and k_0 is a name generated by the requester, if $Priv(k_0) \not\sqsubseteq t_0$ then at each stage w in the run, $Priv(k_0) \not\sqsubseteq t_w$

Proof. Suppose there is a run of DR in which $priv(k_0)$ appears on a message sent over the network. This means, since $Priv(k_0) \not\sqsubseteq t_0$, there is a stage $w > 0$ in the run st

$$Priv(k_0) \not\sqsubseteq t_{w-1} \text{ and } Priv(k_0) \sqsubseteq t_w$$

The event e_w is an event in the set

$$Ev(DR) \equiv Alice : Ev(p_{Alice}) \cup Bob : Ev(p_{Bob}) \cup Spy : Ev(p_{Spy})$$

and by the token game of nets with persistent conditions, is st

$$Priv(k_0) \sqsubseteq e_w^o$$

As can easily be checked, the shape of every *Alice* or *Bob*.

$$e \in Alice : Ev(p_{Alice}) \cup Bob : Ev(p_{Bob})$$

is st

$$Priv(k_0) \not\sqsubseteq e^o$$

The event e_w can therefore only be a Spy event, if $e_w \in Spy : Ev(p_{Spy})$, however by control precedence and the token game , we would find an early stage u in the run, $u < w$ st $priv(k_0) \sqsubseteq t_u$ and therefore a contradiction is reached. □

4.4.4.2 Secrecy Property for the response help intended for the Requester

This theorem concerns the secrecy property for all responses p intended for the requester. It states that all the responses which flow through the network will never be visible for other peers different from the requester.

Theorem 11. Given a run of DR st $X_0 \in Peers(G)$, $p_0 \in Info$, $Priv(k_0) \not\sqsubseteq t_0$ and the run contains a *Bob* event b_1 labeled with action

$$act(b_1) = B : (X_0) : i_0 : j : out\{Rid_0, Pub(k_0), \{\psi, p_0\}_{Pub(k_0)}\}_{Key(X,j)}$$

Where i_0 is a session index, j is an index which belongs to the set $f(X)$, Rid_0 and k_0 are names and $Pub(k_0)$ is a public key associated to the name k_0, and $p_0 \in Info$. Then at every stage w $p_0 \not\sqsubseteq t_w$.

Proof. We show a stronger property such as this

$$Q(p,s,t) \Leftrightarrow \sigma(p_0,t) \subseteq \\ \{\{n_0, Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}, \{\{Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}\}$$

If we can show that at every stage w of the run $Q(p_w, s_w, t_w)$ then clearly $p_0 \notin t_w$ for all stages w in the run. Suppose the opposite statement, that at some stage in the run, property Q does not hold, by freshness clearly $Q(DR, s_0, t_0)$. Let v by well foundedness be the first stage in the run st $\neg Q(p_v, s_v, t_v)$. From the freshness principle it follows

$$a_1 \longrightarrow e_v$$

and from the token game of nets $\{Rid_0, Pub(k_0), \{\psi, p_0\}_{Pub(k_0)}\}_{Key(X,j)} \in \sigma(p_0, t_{v-1})$ (Because messages are persistent in the net). The event e_v is an event in

$$Ev(DR) \equiv Alice : Ev(P_{Alice}) \cup Bob : Ev(P_{Bob}) \cup Spy : Ev(P_{Spy})$$

and from the token game of nets with persistent conditions is st

$$\sigma(p_0, e_v^o - e_{v-1}^o) \not\subseteq \{\{n_0, Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}, \{\{Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}\} \tag{4.7}$$

Clearly e_v can only be an output event since $e_v^o - e_{v-1}^o = \emptyset$ for all input events e. Examining the output events of $Ev(DR)$ we conclude that $e_v \notin Ev(DR)$ reaching a contradiction.

In the following lines we will explore each output event in the protocol in order to verify that the event e_v is different to all of them.

Alice *output events.*

$$act(e_v) = Alice : (X) : j : i : out\,new\,(Rid, k, m)\{Rid, Pub(k), \{m\}_{Pub(k)}\}_{Key(X,i)}$$

Where $X \in Peers(G)$ and so $X \in s_0$, where Rid, m and k are names, $Pub(k)$ is a public key associated to the name k, j is a session index and i is an index which belongs to the set $f(X)$ where $i \in Peers(G)$ and so $i \in s_0$. Property 4.7 and the definition of message surroundings imply that $p_0 \gg \{Rid, Pub(k), \{m\}_{Pub(k)}\}_{Key(X,i)}$. From the freshness property $p_0 \neq Rid$, $p_0 \neq Pub(k)$ and $p_0 \neq m$. Therefore e_v can not be an A event with the above action.

Bob *output events.*

Case Fwd *First output event*

$$act(e_v) = Bob : (X) : j' : j : out\{Rid, Pub(k), \{\psi, p\}_{Pub(k)}\}_{Key(X,j)}$$

Where $X \in Peers(G)$ and so $X \in s_0$, where Rid, m and k are names, $Pub(k)$ is a public key associated to the name k, $p \in info$, j' is a session index and j is an index which belongs to the set $f(X)$ where $j \in Peers(G)$ and so $j \in s_0$. Property 4.7 and the definition of message surroundings imply that $p_0 \gg \{Rid, Pub(k), \{\psi, p\}_{Pub(k)}\}_{Key(X,j)}$. If $p_0 = p$ or $\psi\langle p_0\rangle$ then we reach a contradiction to property 4.7 because from the output principle it follows that $e^o_v - e^o_{v-1} = \{Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,j)}$. Then since property 4.7 must hold, $p_0 = Rid$ or $p_0 = Pub(k)$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

And

$$act(e_u) = Bob : (X) : j' : in\{Rid, Pub(k), \psi\}_{Key(Y,X)}$$

By the token game

$$\{Rid, Pub(k), \psi\}_{Key(Y,X)} \in t_{u-1}$$

and $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ since $\{p_0, Pub(k), \psi\}_{Key(Y,X)} \in \sigma(p_0, t_{u-1})$ or $\{Rid, p_0, \psi\}_{Key(Y,X)} \in \sigma(p_0, t_{u-1})$ and then $\sigma(p_0, t_{u-1}) \not\subseteq \{\{n_0, Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}, \{Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}\}$, a contradiction follows because $u < v$.

Case Fwd *Second output event*

$$act(e_v) = Bob : (X) : j' : out\{n, Rid, Pub(k), \psi\}_{Key(X,Y)}$$

Where $X \in Peers(G)$ and so $X \in s_0$, where n, Rid, m and k are names, $Pub(k)$ is a public key associated to the name k and j' is a session index. Property 4.7 and the definition of message surroundings imply that $p_0 \gg \{n, Rid, Pub(k), \psi\}_{Key(X,Y)}$. If $\psi\langle p_0\rangle$ then we reach a contradiction to property 4.7 because from the output principle it follows that $e^o_v - e^o_{v-1} = \{n_0, Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}$. Property 4.7 must hold then, $p_0 = n$ or $p_0 = Rid$ or $m_0 = Pub(k)$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Bob : (X) : j' : in\{n, Rid, Pub(k), \psi\}_{Key(j,X)}$$

By the token game

$$\{n, Rid, Pub(k), \psi\}_{Key(j,X)} \in t_{u-1}$$

and $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ since $\{p_0, Rid, Pub(k), \psi\}_{Key(j,X)} \in \sigma(p_0, t_{u-1})$ or $\{n, p_0, Pub(k), \psi\}_{Key(j,X)} \in \sigma(p_0, t_{u-1})$ or $\{n, Rid, p_0, \psi\}_{Key(j,X)} \in \sigma(p_0, t_{u-1})$ and then $\sigma(p_0, t_{u-1}) \nsubseteq \{\{n_0, Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}, \{Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}\}$, a contradiction follows because $u < v$.

Case $Triumph$ output event

$$act(e_v) = Bob : (X) : j' : out\, new(n)\, \{n, Rid, Pub(k), \{\psi, p\}_{Pub(k)}\}_{Key(X,Y)}$$

Where $X \in Peers(G)$ and so $X \in s_0$, where n, Rid, m and k are names, $Pub(k)$ is a public key associated to the name k, $p \in info$ and j' is a session index. Property 4.7 and the definition of message surroundings imply that $p_0 \gg \{n, Rid, Pub(k), \{\psi, p\}_{Pub(k)}\}_{Key(X,Y)}$. From the freshness principle, $p_0 \neq n$. If $p = p_0$ or $\psi\langle p_0\rangle$ we reach a contradiction to property 4.7 because from the output principle it follows that $e_v^o - e_{v-1}^o = \{n_0, Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}$. Then since property 4.7 must hold, $p_0 = n$ or $p_0 = Rid$ or $p_0 = Pub(k)$. By control precedence there exists an event e_u in the run st

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Bob : (X) : j' : in\{n, Rid, Pub(k), \psi\}_{Key(Y,X)}$$

By the token game

$$\{n, Rid, Pub(k), \psi\}_{Key(Y,X)} \in t_{u-1}$$

and $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ since $\{p_0, Rid, Pub(k), \psi\}_{Key(Y,X)} \in \sigma(p_0, t_{u-1})$ or $\{n, p_0, Pub(k), \psi\}_{Key(Y,X)} \in \sigma(p_0, t_{u-1})$ or $\{n, Rid, p_0, \psi\}_{Key(Y,X)} \in \sigma(p_0, t_{u-1})$ and then $\sigma(p_0, t_{u-1}) \nsubseteq \{\{n_0, Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}, \{Rid_0, Pub(k_0), \psi\langle p_0\rangle\}_{Key(X,Y)}\}$, a contradiction follows because $u < v$.

Spy *output events* An assumption of the theorem is that the private key of the requester is not leaked, meaning that $Priv(k) \nsubseteq t_0$. At every stage w in the run $Priv(k) \nsubseteq t_w$. Since this there is no possible way for a spy to reach p_0, e_v is not a spy event. □

4.4.5 Integrity Proofs in DR

The requester guarantees the integrity of the message it will receive, by adding in the first layer, a fresh name m, encrypted with a new public key $Pub(s)$. This value should be kept inside the message in order to be recognized. Since the name m is included in the message in the same way as p, and we have already proved the secrecy property for the response information p in section 4.4.4.2, we can state that m is kept as a secret along the protocol. In this case, we can ensure that nobody different from the requester has access to m. Since every helper must add some information to the message, and the only way to keep the m value is maintaining the already received contents, the helper must add its new data and cover the whole message with a new encryption layer generated with $Pub(s)$. Then, if it can be guaranteed that the name m persists in the message, and this nonce is never leaked (already verified), the integrity of the message, is never harmed.

This integrity property is verified by presenting a property which states that every message intended for the requester has the same structure which indicates that the nonce m is always present, and as we said, if m is kept as a secret, the integrity of the message is ensured. In order to verify this property, each output event occurring in the protocol must be verified, to ensure that there is no message intended for the requester, which appear without nonce m.

4.4.6 Integrity Property for the messages intended for the Requester

This theorem states that the same fresh name m will always appear in the same message identified with a request id Rid. This property among with the secrecy property for value m will ensure the integrity of the message.

Theorem 12. Given a run of DR, $X_0 \in Peers(G)$, $Priv(k_0) \not\sqsubseteq t_0$, and the run contains an $Alice$ event a_1 labelled with action

$$act(a_1) = Alice : (X_0) : i_0 : i : out\{Rid_0, Pub(k_0), \{m_0\}_{Pub(k_0)}\}_{Key(X,i)}$$

Where i_0 is a session index, i is an index which belongs to the set $f(X)$, Rid_0, m_0 and k_0 are names and $Pub(k_0)$ is a public key associated to the name k_0, then at every stage w the integrity of the message will be maintained.

Proof. We show the formalized proof in the following property:

$$Q(p, s, t, m_0) \Leftrightarrow \forall M \in \sigma(Rid_0, t) \,.\, M \sqsubseteq \{n_0, Rid_0, Pub(k_0), \psi\langle m_0\rangle[m_0/m]\}_{Key(X,Y)}$$

If we can show that at every stage w of the run $Q(p_w, s_w, t_w, m_0)$ then clearly the integrity of the message is maintained along all stages w in the run. Suppose the contrary, suppose that at some stage in the run, property Q does not hold, by freshness clearly $Q(DR, s_0, t_0, m_0)$. Let v by well foundedness be the first stage in the run st $\neg Q(p_v, s_v, t_v, m_0)$. From the freshness principle it follows

$$a_1 \longrightarrow e_v$$

and from the token game of nets $\{Rid_0, Pub(k_0), \{m_0\}_{Pub(k_0)}\}_{Key(X,i)} \in \sigma(Rid_0, t_{v-1})$ (Because messages are persistent in the net). The event e_v is an event in

$$Ev(DR) \equiv Alice : Ev(P_{Alice}) \cup Bob : Ev(P_{Bob}) \cup Spy : Ev(P_{Spy})$$

and from the token game of nets with persistent conditions is st

$$\sigma(Rid_0, e_v^o - e_{v-1}^o) \not\sqsubseteq \{n_0, Rid_0, Pub(k_0), \psi\langle m_0\rangle[m_0/m]\}_{Key(X,Y)} \wedge \forall m_i \in \sigma(Rid_0, e_v^o - e_{v-1}^o)\,,\; m_0 \gg m_i \tag{4.8}$$

Clearly e_v can only be an output event since $e_v^o - e_{v-1}^o = \emptyset$ for all input events e. Examining the output events of $Ev(DR)$, we conclude that $e_v \notin Ev(DR)$ reaching a contradiction.

Since we are analyzing the integrity of messages intended for the requester, we will take a look at specific output processes where a particular message identified by a Request id Rid_0 occurs. (Where $Rid = Rid_0$). We explore these events in order to verify that the event e_v is different to all of them.

Alice *output events.*

$$act(e_v) = Alice : (X) : j : i : out\,new\,(Rid, k, m)\{Rid, Pub(k), \{m\}_{Pub(k)}\}_{Key(X,i)}$$

Where $X \in Peers(G)$ and so $X \in s_0$, where Rid, m and k are names, $Pub(k)$ is a public key associated to the name k, j is a session index and i is an index which belongs to the set $f(X)$ where $i \in Peers(G)$ and so $i \in s_0$. Property 4.8 and the definition of message surroundings imply that $m_0 \gg \{Rid, Pub(k), \{m\}_{Pub(k)}\}_{Key(X,i)}$. Since $Rid = Rid_0$ then $m_0 \neq Rid$. And from the freshness property $m_0 \neq Pub(k)$. Then, if $m_0 = m$ then we reach a contradiction to property 4.8 because from the output principle it follows that $e_v^o - e_{v-1}^o = \{Rid_0, Pub(k_0), \psi\langle m_0\rangle[m_0/m]\}_{Key(X,i)}$. Therefore e_v can not be an A event with the above action.

Bob *output events.*

Case Fwd First output event

$$act(e_v) = Bob : (X) : j' : j : out\,\{Rid, Pub(k), \{\psi, p\}_{Pub(k)}\}_{Key(X,j)}$$

Where $X \in Peers(G)$ and so $X \in s_0$, where Rid, m and k are names, $Pub(k)$ is a public key associated to the name k, $p \in info$, j' is a session index and j is an index which belongs to the set $f(X)$ where $j \in Peers(G)$ and so $j \in s_0$.Property 4.8 and the definition of message surroundings imply that $m_0 \gg \{Rid, Pub(k), \{\psi, p\}_{Pub(k)}\}_{Key(X,j)}$. $Rid = Rid_0$ then $m_0 \neq Rid$. Since $p \in Info$ and so $p \in s_0$ from the freshness principle it follows that $m_0 \neq p$. If $m_0 = m$ then we reach a contradiction to property 4.8 because from the output principle it follows that $e_v^o - e_{v-1}^o = \{Rid_0, Pub(k_0), \psi\langle m_0\rangle[m_0/m]\}_{Key(X,j)}$. Then since property 4.8 must hold, $m_0 = Pub(k)$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Bob : (X) : j' : in\{Rid, m_0, \psi\}_{Key(Y,X)}$$

By the token game

$$\{Rid, m_0, \psi\}_{Key(Y,X)} \in t_{u-1}$$

where $n_0 \neq Pub(k_0)$ and so $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ which is a contradiction because $u < v$.

Case Fwd Second output event

$$act(e_v) = Bob : (X) : j' : out\,\{n, Rid, Pub(k), \psi\}_{Key(X,Y)}$$

Where $X \in Peers(G)$ and so $X \in s_0$, where n, Rid, m and k are names, $Pub(k)$ is a public key associated to the name k and j' is a session index. Property 4.8 and the definition of message surroundings imply that $m_0 \gg \{n, Rid, Pub(k), \psi\}_{Key(X,Y)}$. $Rid = Rid_0$ then $m_0 \neq Rid$. Since $p \in Info$ and so $p \in s_0$ from the freshness principle it follows that $m_0 \neq p$. If $m_0 = m$ then we reach a contradiction to property 4.8 because from the output principle it follows that $e^o_v - e^o_{v-1} = \{n_0, Rid_0, Pub(k_0), \psi\langle m_0\rangle[m_0/m]\}_{Key(X,Y)}$. Property 4.8 must hold then, $m_0 = n$ or $m_0 = Pub(k)$. By control precedence there exists an event e_u in the run st.

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Bob : (X) : j' : in\{n, Rid, Pub(k), \psi\}_{Key(j,X)}$$

By the token game

$$\{n, Rid, Pub(k), \psi\}_{Key(j,X)} \in t_{u-1}$$

and $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ since $\{m_0, Rid, Pub(k), \psi\}_{Key(j,X)} \in \sigma(m_0, t_{u-1})$ or $\{n, Rid, m_0, \psi\}_{Key(j,X)} \in \sigma(m_0, t_{u-1})$ and then $\sigma(m_0, t_{u-1}) \not\sqsubseteq \{n_0, Rid_0, Pub(k_0), \psi[m_0/m]\}_{Key(X,Y)}$, a contradiction follows because $u < v$.

Case $Triumph$ output event

$$act(e_v) = Bob : (X) : j' : out\,new(n)\,\{n, Rid, Pub(k), \{\psi, p\}_{Pub(k)}\}_{Key(X,Y)}$$

Where $X \in Peers(G)$ and so $X \in s_0$, where n, Rid, m and k are names, $Pub(k)$ is a public key associated to the name k, $p \in info$ and j' is a session index. Property 4.8 and the definition of message surroundings imply that $m_0 \gg \{n, Rid, Pub(k), \{\psi, p\}_{Pub(k)}\}_{Key(X,Y)}$. $Rid = Rid_0$ then $m_0 \neq Rid$. From the freshness principle, $m_0 \neq n$. Since $p \in Info$ and so $p \in s_0$ from the freshness principle it follows that $m_0 \neq p$. If $m_0 = m$ then we reach a contradiction to property 4.8 because from the output principle it follows that $e^o_v - e^o_{v-1} = \{n_0, Rid_0, Pub(k_0), \psi\langle m_0\rangle[m_0/m]\}_{Key(X,Y)}$. Then since property 4.8 must hold, $m_0 = n$ or $m_0 = Pub(k)$. By control precedence there exists an event e_u in the run st

$$e_u \longrightarrow e_v$$

and

$$act(e_u) = Bob : (X) : j' : in\{n, Rid, Pub(k), \psi\}_{Key(Y,X)}$$

By the token game

$$\{n, Rid, Pub(k), \psi\}_{Key(Y,X)} \in t_{u-1}$$

and $\neg Q(p_{u-1}, s_{u-1}, t_{u-1})$ since $\{m_0, Rid, Pub(k), \psi\}_{Key(Y,X)} \in \sigma(m_0, t_{u-1})$ or $\{n, Rid, m_0, \psi\}_{Key(Y,X)}$ $\in \sigma(m_0, t_{u-1})$ and then $\sigma(m_0, t_{u-1}) \not\sqsubseteq \{n_0, Rid_0, Pub(k_0),\ \psi\langle m_0\rangle[m_0/m]\}_{Key(X,Y)}$, a contradiction follows because $u < v$.

Spy *output events* Since we have proved before that the private key $Priv(k)$ is never leaked, we can guarantee that no Spy can ever change the contents of the messages, then e_v is not a Spy event. □

4.5 Discussion

This chapter presents two main ideas we want to extend, the first relies on the modeling and specification of a new set of constructions closely related to concurrency models. Although these are not new ideas and are present in other process calculi such as [MPW89, AG97a, AF01, Hoa83, Car99], a pure inclusion of these kind of tools in SPL presents serious difficulties according to the inherent model of persistent networks. Therefore, by using the nominality of this calculus together with strong encryption mechanisms, this kind of constructions can be emulated without any intrusive changes to SPL operational semantics. Hence, providing a set of encodings allows a clean and straight-forward translation between a broader subset of protocols models in different concurrency models mentioned before and SPL. However, a strong relation concerning the expressiveness is necessary to achieve a complete translation within them. Previous works establishing strong relations between lambda-calculus[Chu51] and the π calculus, and between persistent and non-persistent languages are presented by means of encodings[SW01, GSV04, PSVV04]. Relying on this concepts, an interesting strand of research could involve an encoding from SPL to the asynchronous π calculus in such a way that every calculus π-reducible can be translated to SPL in order to use its simple but powerful reasoning techniques.

Our second contribution we want to stand out relates to the formalization and proof of new security properties using a process calculi. In particular, we have considered Integrity as one of the essential properties in order to guarantee the security of the system, particularly in applications where mobility involves extensibility of services, resources or functionality. Security information technologies have presented different approaches to tackle it, involving every one of the levels in information security, from ACID control mechanisms [SK86], to security protocols [ZS00] and policies [BF03]. However, reasoning techniques provided by process calculi, in particular SPL, brings the necessary flexibility to construct a powerful framework to prove different security properties, a clear advantage from specific-driven models.

4.6 Summary

This chapter was specially devoted to exploring SPL language in other contexts, with a subset of protocols of P2P systems specially designed to deal with issues in collaborative computing. In this way, we use the Friends Troubleshooting Network protocol as a well grounded example where contribution among peers its critical for the correctness of the protocol, considering a number of aspects where security comes of the essence.

With this model, we have constructed a set of encodings that allow a protocol designer to construct models with close resemblance to widely known models such as Spi calculus and CSP, broadening the elements provided in SPL for a more straight-forward design of protocols closely to implementation stage. This encodings were show in the compositional model of FTN protocol using SPL.

The last, but no least important task, was to abstracting the functionality of FTN protocol and based on concepts of multi-layered encryption systems, propose a new model of dynamic reconfiguration protocol, simpler in its behavior, but efficient in the number of process involved between peers. To validate the Dynamic Reconfiguration protocol, we prove its correctness by means of the use of SPL reasoning techniques to formally two important security properties, such as secrecy and integrity.

5 Concluding Remarks

This chapter aims to relate the current research results with other approaches of information security, stating the principal conclusions derived from achieved results, and pointing to several directions where this research can be extended.

5.1 Related Work

Information Security is a well studied area, in such a way that a wide variety of formalisms have been developed to overcome the risks exposed in chapter 2. In this section we compare these formalisms with the ones used along this thesis, firstly comparing other frameworks for security analysis, and secondly studying previous works in formal models for P2P systems.

Approaches for Security Analysis

- *State-exploration Models:* These techniques are focused in the exhaustive exploration of every possible interaction of concurrent processes in order to find at least one state where the invariant is not fulfilled, showing interesting results breaking protocols considered as secure with other techniques[Ros94, Low96, Low97, MCJ97], with existent implementations [Low97, MCJ97]. However, the nature of interactions between processes leads us to a problem known as *The State Explosion Problem [Kot03]*, imposing limits on the size and complexity of the protocols in terms of processes involved, making really hard to express protocols with infinite behavior such as those used in P2P systems. However, improvements in space-exploration search algorithms, satisfacibility models [SRP91] and graph theory seems promising in order to consider only reachable states [QvRDC06].
- *Logic Models :* Probably the widest used technique, these models define the knowledge of a system in terms of beliefs of each one of the agents, providing a set of rules to denote the evolution of the knowledge in the system [BAN96]. Each model deals in a different way with the information, ranging from agent-driven models [BAN96, AG97a, AG97b, AG99, AF01] to network-driven models [Pau97, Cra03]. The properties are proved by means of using rules present in the models in order to find a reduction that violates the invariant established, using logic programming as a powerful tool capable of representing and

implementing such type of reasonings with satisfactory results [KW96, Mea92, Mea96]. SPL has a strong resemblance with this approach, basing its transition semantics on this models. However, one of the virtues in this models is the ability to relate two different protocols in order to find similar behaviors. This is an interesting strand of research, which eases the work required to prove the security of some protocols, only by reducing to well-known examples proved from the scratch.

- *Temporal Logic:* By combining both temporal and first order logics, frameworks in which systems requiring both dynamic and informational aspects relating to knowledge can be described [DGFvdH04, HT96, JWM95]. This is particularly important in security protocols, where one wants to ensure that certain knowledge is obtained over time or, at least, the ignorance of potential intruders persists over the whole run of the protocol. These logics have advantages of a well-defined semantics, generating a framework more formal than the previous models studied. However, these classes of frameworks are cumbersome, needing long proofs for even simple protocols [JWM95], and are rather complex for a suitable implementation; in previous research, the complexity of the model itself with proofs of "secrecy-temporality" are shown to be undecidable[HT96]. Relating to SPL model, the event-based approach has been successfully automated by implementing a complete framework called $\chi - Spaces$ [Mil02] which is closely tied to SPL semantics, providing an efficient way to model, simulate and implement security protocols

- *Constraint-based Models:* One of the novels ideas in information security address the use of constraint programming (CP [MS98]) as a suitable tool to model security protocols and policies. Constraint Solving is an emerging software technology for declarative description and resolution of large problems. In this approach Bistarelli models the system as a constraint satisfaction problem where agents are represented by variables bounded with domains that denote the messages present in the network. The interaction between agents are modelled as a set of constraints that acts over the variables, and the proofs are simply modelled in the resolution of the constraint problem, verifying cases related to confidentiality of a message [BB01] or security-policies [BF03]. One of the most relevant characteristics of these approach is the use of a monotonic store of constraints where partial information over domain variables is increased by *tell* operations. This inference mechanism resembles the monotonic space of messages present in SPL. However the models maintain strong differences between them: The first of them relates to proof analysis, meanwhile protocols in the approach of constraint programming are proved in a fully automated way, proofs in SPL have to be manually defined. The second difference relates to the properties itself, SPL provides a strong set of proof principles appropriate for the definition and verification of a wide variety of security properties, meanwhile proofs in CP have to be defined as a derivation of a property related to the privacy levels of a system.

Formal models for P2P systems Although the use of process calculi for the analysis of security aspects is a topic well studied in the literature, including works in the π

and Spi calculus [SW01, Mil99, AG99], CSP process algebra [Sch96c, RSG$^+$01] and ambient calculus [Car99]. To our knowledge, little work has been done in security analysis of P2P protocols using Process Calculi. In particular, the project Pepito [HS02] has started efforts in verification of properties using CCS variants in static versions of P2P protocols [BNAG04], in particular, correctness properties. Other analysis have been made for specific P2P functionalities, like quantitative analysis [SL04] and trust reputation models [SL03, AD01, GJA03, KSGM03]. However, to our knowledge, this is the first formal attempt using process calculi to model and reason about security properties in P2P protocols.

5.2 Conclusions

1. The use of process calculi as tools to model, analyze and verify communication concurrent systems, allows us to formalize any kind of communication protocols leaving aside technical details. Transforming complex distributed algorithms into abstract models syntactically close to their descriptions in pseudocode. Enabling a detailed description of their behaviors by means of several mechanisms such as equivalences representing actions by which each component in the system evolves. In particular process calculi concerned to security, allows us to model security protocols using their inherent cryptographic primitives, as well as to enable verification of security properties by using their own different operational semantics.

2. The use of the SPL calculus let us model several processes involved in popular real life protocols, such as those involved in P2P systems, without loosing dependencies among them, in order to verify security properties along all their runs. In this way, properties essential for P2P communications protocols can easily be verified. We demonstrate this by modeling an analyzing two protocol examples related to the most representative P2P systems, where collaborative processing and sharing of information have become critical tasks associated to security.

3. After deeply analyzing several crucial properties an important protocol such as MUTE must fulfill, several failures with respect to security attacks behind different kind of saboteurs were stated. In this way, taking in count such failures behind a more powerful attacker, such as the one which can impersonate a trusted user inside the network, we add a new component to the protocol structure, as well as other partial modifications in the communication model, to prevent an important attack known as *the middle man attack*. The inclusion of a file controller and several modifications to the MUTE protocol, give life to the modified MUTE protocol presented in 3.6 which can avoid these kind of attacks inside the network.

4. A very important contribution within this work, regards to the inclusion of several features present in other different protocols, into another protocol which presented some failures with respect to several attacks, so it could be modelled and verified under the SPL model, in order to fulfill each property established by its optimal scheme. This can

be easily seen in chapter 4, where we improve the simple FTN protocol, by developing a new protocol, the Dynamic Reconfiguration protocol, with the same functionality presented in FTN, but with a very important feature, known as a layer encryption, used in industrial and military protocols [GRS99], which enables the achievement of a new property denoted as message integrity.

5. The underpinning theory by which SPL relies on, among with its flexible and intuitive proof techniques, enables not just proving the security properties already verified in other works, but the exaltation of its generality, since by means of subtle modifications to the general proof structure, one can verify different important properties never proved. One of these is the integrity property, verified in our new Dynamic Reconfiguration Protocol defined in 4.4.

6. We bear witness of the flexibility and generality of SPL reasoning techniques, since in a relatively simple way, without major relevant changes in the general proof structure established in SPL, we could prove other kinds of protocols never verified or even modelled. Then, by means of case studies in chapters 3 and 4 we can conclude that even though SPL is a very simple security language, it presents a very high level of polyvalence with respect to modeling and verifying several type of protocols.

7. Albeit the SPL protocol language presents an expressive and powerful semantics, real world protocols need a broader set of constructions for being expressed accurately. We relate a set of these constructions with other process calculi existent, and according to its relevance, we use the syntactic set present in SPL to model encodings, supplying protocol designers a wider set of constructions without intrusive extensions of the calculus. In particular, we enable a much more clear and precise security communication protocol model for the fixed FTN protocol defined 4.1.2, where a mutability construction is needed, as well as a set of constructions which could represent notions such as the non-deterministic choice and sequential composition.

5.3 Future Work

The following ideas emerges as directions for future work:

5.3.1 Local reasoning in SPL

In chapter 4 we have seen the model of FTN we discuss about the inclusion of local computation for processes. In other process calculi, this is an easy task with the inclusion of functions. In SPL we can achieved this by using message exchanges and private keys in every execution of a process. In this case, the function will behave as follows: giving a vector of attribute-value messages, the function will insert new values for each attribute and generate new tuples. More formally, the specification is presented below:

$$
\begin{aligned}
& fun(\vec{x}) \rightarrow \vec{x}' \cup \vec{w} \\
\text{Where:} & \\
& \vec{x} : \langle\langle a_1, \vec{v}_1\rangle, \langle a_2, \vec{v}_2\rangle, \ldots, \langle a_n, \vec{v}_n\rangle\rangle \\
& \vec{x}' : \langle\langle a_1, \vec{v}'_1\rangle, \langle a_2, \vec{v}'_2\rangle, \ldots, \langle a_n, \vec{v}'_n\rangle\rangle \\
& \vec{w} : \langle\langle b_1, \vec{u}_1\rangle, \langle b_2, \vec{u}_2\rangle, \ldots, \langle b_n, \vec{u}_n\rangle\rangle \\
\text{And} & \\
& \parallel \vec{v}_i \parallel \leq \parallel \vec{v}'_i \parallel \wedge \forall x \in \vec{v}_i \mid x \in \vec{v}'_i
\end{aligned}
$$

To correctly model this function, we specify it with three processes, where the basic process $h(w,k)$ simply takes a single attribute-value tuple and represents the local computing of values with an insertion of a new vector of nonces to the tuple, sending it with a previously received key and a new value that guarantees the freshness of the message. The next process $g(\vec{x}, l)$ splits the message into attribute-value tuples, the local computation function for each tuple, checks the integrity of the response, and sends the tuple for further use. Finally the process $fun(A, j, \vec{x})$ only generates a key to use in previous processes, sequentially receives every submessage of $\vec{x}$ and includes the new tuples in the request, sending the results over a public key j of the agent A.

Definition 24. Let $S(\vec{x})$ be the subset of messages of a vector $\vec{x}$ composed by attribute-value tuples $w_i = (a_i, \vec{v}_i)$. Let x, y the composition of messages x and y; and $U = \Pi_{i \in \{1 \ldots k\}} w_i$ the creation of a message U composed by every message w_i that belongs to the indexed set $\{1 \ldots k\}$.

In this way, a function that includes a new vector of attribute-value tuples from a vector previously determined can be seen as follows:

$$
\begin{aligned}
fun(A, j, \vec{x}) &\triangleq \; out\, new\, l\, \{l\}_{Pub(A)} .g(\vec{x}, l).(\|_{i \in S(\vec{x})}\; in\; \{w_i\}_{Pub(l)}).out\, new\, \vec{x'}\, \{U, \vec{x'}\}_j \\
g(\vec{x}, l) &\triangleq \; \|_{w_i \in S(\vec{x})}\; out\, new\, k_i\, \{k_i\}_{Pub(k_i)} h(w_i, k_i).in\, \{y, a_i, (\vec{v_i}, \vec{v'_i})\}_{Pub(k_i)}.out\, \{a_i, (\vec{v_i}, \vec{v'_i})\}_{Pub(l)} \\
h(w, k) &\triangleq \; out\, new(y, \vec{v'})\{y, a, (v, \vec{v'})\}_{Pub(k)}
\end{aligned}
$$

It is clear that a modeling of such an easy function like $fun(\vec{x})$ is not a trivial task in SPL. The inclusion of even local names only concerning to an agent must be modelled as output actions with nonces to guarantee the freshness of the message, with encryptions to ensure that the information remains private from eavesdropping. From an practical point of view, encryptions are not an economic process, involving common task such as factoring and decompression; and the approach of SPL to model local computations, although possible, is complex and useless. From works like CCS [Mil95], π calculus [MPW89] or even ntcc [PV01, NPV02] a notion of observational processes is present. We think that an inclusion of local computations on SPL reasoning techniques can provide elements for the reasoning of other kind of processes, even for new threats, like dictionary and guessing attacks [Low04, RS98]. Some directions of this works include the integration of SPL with dynamic and mobile classes of Petri nets [AB96], and the inclusion of CCP [SRP91] reasoning techniques.

5.3.2 New models of adversaries

Recent works from the literature shows that although the Dolev-Yao adversary model presented in almost all existing logics for security, is too restrictive in the power of an attacker, assuming that an agent cannot infer information about message structure or knowledge about the protocol being used [PH02]. Some works extending attacker capabilities demonstrate that attackers with more knowledge from the systems can corrupt protocols previously proved as secure systems [Low95]. Although in this document we expand the attacker model with intrusion capabilities, some assumptions can be proved more explicitly with an stronger model of an attacker, including notions of *algorithmic knowledge [FHVM95]* and probability. An interesting strand of research can be derived from this works, adapting the model of attackers in SPL with models that explicit adversaries limitations.

5.3.3 Relating Security Models

As we have seen in section 5.1, SPL is close to a number of logic approaches such as the Asynchronous π calculus. In another way, several works demonstrate that the expressiveness of other process calculi used for security such as the Spi and Applied π calculus can be encoded into π [BFH04]. Others demonstrate how the behavior presented in persistent output π calculus can be encoded in π calculus [PSVV04]. We believe that works encoding SPL behavior to π calculus with persistent conditions can close the gap between these different models, allowing translations between languages, enabling the use of SPL reasoning techniques for verifying Spi-modelled protocols, as well as allowing equivalences between SPL processes, provided by techniques such as bisimulations or congruences inherent to the Spi calculus.

5.3.4 Protocol Implementation

We strongly believe that efficient implementations of both DR and ModMUTE protocols can be suitable in the meantime as an useful tool to achieve and compare our theoretical results with the practicality of real world systems. We first propose the use of $\chi - Spaces$ as an interesting framework for the development of this protocols, due to its strong bow with SPL. Since $\chi - Spaces$ is an automatic framework based on SPL, there is no much problems with the usual gaps between the formal model of the protocol and its implementation. Similarly, it will be very interesting to implement the proposed SPL encodings of new constructions in $\chi - Spaces$, to enable much clear and precise implementations of several different protocols.

Bibliography

[AB96] Andrea Asperti and Nadi Busi. Mobile Petri Nets. Technical Report UBLCS-96-10, University of Bologna, 1996.

[Aba00] Martin Abadi. Security protocols and their properties. In F.L. Bauer and R. Steinbrueggen, editors, *Foundations of Secure Computation*, 20th Int. Summer School, Marktoberdorf, Germany, pages 39–60. IOS Press, 2000.

[AD01] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In Henrique Paques, Ling Liu, and David Grossman, editors, *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM01)*, pages 310–317. ACM Press, 2001.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 104–115, New York, NY, USA, 2001. ACM Press.

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

[AG97b] Martín Abadi and Andrew D. Gordon. Reasoning about cryptographic protocols in the spi calculus. In *CONCUR'97: Concurrency Theory*, volume 1243, pages 59–73. Springer-Verlag, Berlin Germany, 1997.

[AG99] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[ALR05] Andrés Aristizábal, Hugo A. López, and Camilo Rueda. Using a declarative process language for P2P protocols. *The Association for Logic Programming Newsletter Digest*, 18(4), November 2005.

[ASL00] Ross Andreson, Frank Stajano, and Jong-Hyeon Lee. Security policies. In *Advances in Computers*, 2000.

[BAN96] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication, from proceedings of the royal society, volume 426, number 1871, 1989. In

William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996. 1996.

[BB01] Giampaolo Bella and Stefano Bistarelli. Soft constraints for security protocol analysis: Confidentiality. In *PADL '01: Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages*, pages 108–122, London, UK, 2001. Springer-Verlag.

[BC02] Alexander Bockmayr and Arnaud Courtois. Using hybrid concurrent constraint programming to model dynamic biological systems. In Peter J. Stuckey, editor, *ICLP*, volume 2401 of *Lecture Notes in Computer Science*, pages 85–99. Springer, 2002.

[BF03] Stefano Bistarelli and S. Foley. Analysis of integrity policies using soft constraints. In *Proceedings of IEEE Workshop Policies for Distributed Systems and Networks*, pages 77–80, 2003.

[BFH04] Anders Bloch, Morten V. Frederiksen, and Bjørn Haagensen. The applied π calculus: Type systems and expressiveness. Master's thesis, Aalborg University, 2004.

[BMWZ05] Matthias Bender, Sebastian Michel, Gerhard Weikum, and Christian Zimmer. The minerva project: Database selection in the context of p2p search. In *BTW*, pages 125–144, 2005.

[BNAG04] Johannes Borgström, Uwe Nestmann, Luc Onana Alima, and Dilian Gurov. Verifying a structured peer-to-peer overlay network: The static case. In TBD, editor, *Proc. Global Computing 2004*, volume TBD of *Lecture Notes in Computer Science*, page TBD. Springer, 2004.

[BP01] G. Bella and L. C. Paulson. Mechanical proofs about a non-repudiation protocol. In R. J. Boulton and P. B. Jackson, editors, *Proceedings of the 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2001)*, volume 2152 of *Lecture Notes in Computer Science*, pages 91–104, Edinburgh, Scotland, UK, September 2001. Springer-Verlag.

[BS04] S. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. Technical report, Computer Science Department,Columbia University, September 2004.

[Car99] Luca Cardelli. Mobilility and security. In *In F. L. Bauer and R. Steinbrueggen, editors, Foundations of Secure Computation, NATO Science Series, pages 3-37. IOS Press, 2000.*, 1999.

[Car04] Luca Cardelli. Brane calculi. In Vincent Danos and Vincent Schachter, editors, *CMSB*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2004.

[CCM02] Mario José Cáccamo, Federico Crazzolara, and Giuseppe Milicia. The ISO 5-pass authentication in χ-Spaces. In Youngsong Mun and Hamid R. Arabnia, editors, *Proceedings of the Security and Management Conference (SAM'02)*, pages 490–495, Las Vegas, Nevada, USA, June 2002. CSREA Press.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98*. Springer-Verlag, Berlin Germany, 1998.

[CHP71] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent control with readers and writers. *Commun. ACM*, 14(10):667–668, 1971.

[Chu51] A. Church. *The Calculi of Lambda-Conversion*, volume 6 of *Annals of Mathematical Studies*. Princeton University Press, Princeton, 1951. (second printing, first appeared 1941).

[CoI05] National White Collar Crime Center and Federal Bureau of Investigation. Ic3 2004 internet fraud-crime report, 2005.

[Cra03] Federico Crazzolara. Language, semantics, and methods for security protocols. Doctoral Dissertation DS-03-4, brics, daimi, May 2003. PhD thesis. xii+160.

[CW01] Federico Crazzolara and Glynn Winskel. Events in security protocols. In *ACM Conference on Computer and Communications Security*, pages 96–105, 2001.

[DGFvdH04] Clare Dixon, Mari-Carmen Fernandez Gago, Michael Fisher, and Wiebe van der Hoek. Using temporal logics of knowledge in the formal verification of security protocols. In *TIME '04: Proceedings of the 11th International Symposium on Temporal Representation and Reasoning (TIME'04)*, pages 148–151, Washington, DC, USA, 2004. IEEE Computer Society.

[DGOR04] Juan Francisco Díaz, Gustavo Gutierrez, Carlos Alberto Olarte, and Camilo Rueda. Cre2: A cp application for reconfiguring a power distribution network for power losses reduction. In *CP*, pages 813–814, 2004.

[DS04] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization.* MIT Press, July 2004.

[DY81] Danny Dolev and Andrew C. Yao. On the security of public key protocols. Technical report, Dept. of Computer Science, Stanford University, Stanford, CA, USA, 1981.

[Ese02] A. Esenther. Instant co-browsing: Lightweight real-time collaborative web browsing, 2002.

[FG01] Riccardo Focardi and Roberto Gorrieri. Classification of security properties (part i: Information flow). In *FOSAD '00: Revised versions of lectures given* *during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design*, pages 331–396, London, UK, 2001. Springer-Verlag.

[FHVM95] Ronald Fagin, Joseph Y. Halpern, Moshe Y. Vardi, and Yoram Moses. *Reasoning about knowledge.* MIT Press, Cambridge, MA, USA, 1995.

[GJA03] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks. In *NOSSDAV 2003.*, June 2003.

[GK03] Nathaniel S. Good and Aaron Krekelberg. Usability and privacy: a study of kazaa p2p file-sharing. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 137–144, New York, NY, USA, 2003. ACM Press.

[Gol99] Dieter Gollmann. *Computer security.* John Wiley & Sons, Inc., New York, NY, USA, 1999.

[GPR05] Julian Gutiérrez, Jorge Andrés Pérez, and Camilo Rueda. Time, nondeterminism and constraints in modeling and verifying biological systems. Unpublished, September 2005.

[GRS99] D. Goldschlag, M. Reed, and P. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM (USA)*, 42(2):39–41, 1999.

[GSV04] Pablo Giambiagi, Gerardo Schneider, and Frank D. Valencia. On the expressiveness of infinite behavior and name scoping in process calculi. In Igor Walukiewicz, editor, *FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*, pages 226–240. Springer, 2004.

[Hoa83] C. A. R. Hoare. Communicating Sequential Processes. *Commun. ACM*, 26(1):100–106, 1983.

[HS02] Seif Haridi and Thom Sjöland. Pepito - PEer-to-Peer: Implementation and TheOry, 2002.

[HT96] Nevin Heintze and J. D. Tygar. A model for secure protocols and their compositions. *Software Engineering*, 22(1):16–30, 1996.

[Hut01] Michael R A Huth. *Secure Communicating Systems.* Cambridge University Press, first edition, 2001.

[HWB05] Qiang Huang, Helen J. Wang, and Nikita Borisov. Privacy-preserving friends troubleshooting network. *ISOC NDSS, San Diego, CA.*, 2005.

[JWM95] Iii Gray J. W. and J. McLean. Using temporal logic to specify and verify cryptographic protocols. In *CSFW '95: Proceedings of the The Eighth IEEE*

Computer Security Foundations Workshop (CSFW '95), page 108, Washington, DC, USA, 1995. IEEE Computer Society.

[KD03] J. Krivine and V. Danos. Formal molecular biology done in CCS-R. In *Bio-Concur 2003, Workshop on Concurrent Models in Molecular Biology*, 2003.

[Kot03] Martin Koth. The State Explosion Problem, August 2003.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[KW96] Darrell Kindred and Jeannette Wing. Fast, automatic checking of security protocols. In *Proccedings of 2 nd Usenix Workshop on Electronic Commerce*, pages 41–52, 1996.

[Lai92] X. Lai. *On the Design and Security of Block Ciphers.* Konstanz, Hartung-Gorre, Germany, 1992.

[Low95] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[Low96] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. *Software - Concepts and Tools*, 17(3):93–102, 1996.

[Low97] Gavin Lowe. Casper: A compiler for the analysis of security protocols. In *CSFW, 10th Computer Security Foundations Workshop (CSFW '97)*, pages 18–30. IEEE Computer Society, 1997.

[Low04] Gavin Lowe. Analysing protocol subject to guessing attacks. *Journal of Computer Security*, 12(1):83–98, 2004.

[MCJ97] W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. Technical report, Carnegie Mellon University, 1997.

[Mea92] Catherine Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1), 1992.

[Mea96] Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[Mil95] Robin Milner. *Communication and concurrency.* Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1995.

[Mil99] Robin Milner. *Communicating and Mobile systems. The Pi Calculus.* Cambridge University Press, 1999.

[Mil02] Giuseppe Milicia. χ-Spaces: Programming Security Protocols. In *Proceedings of the 14th Nordic Workshop on Programming Theory (NWPT'02)*, November 2002.

[MKL$^+$02] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, March 2002.

[MPW89] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. Technical Report -86, 1989.

[MS98] Kim Marriott and Peter J. Stuckey. *Introduction to Constraint Logic Programming.* MIT Press, Cambridge, MA, USA, 1998.

[NPV02] Mogens Nielsen, Catuscia Palamidessi, and Frank Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. *Nordic J. of Computing*, 2002.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[Pau97] Lawrence C. Paulson. Proving properties of security protocols by induction. In *10th Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, 1997.

[Per96] Charles Perkins. IP Mobility Support - RFC2002. IETF RFC Publication, 1996.

[Pet77] James L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, 1977.

[Pfl96] Charles Pfleeger. *Security In Computing.* Prentice Hall, second edition, 1996.

[PH02] R. Pucella and J. Halpern. Modeling adversaries in a logic for security protocol analysis. In *Formal Aspects of Security, 2002 (FASec '02)*, 2002.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical report, University of Aarhus, 1981.

[PR99] N. De Palma and B. Riveill. Dynamic reconfiguration of agent-based applications. In *European Research Seminar on Advances in Distributed systems (ERSADS'99), Madeira, Portugal*, April 1999.

[PSVV04] Catuscia Palamidessi, Vijay Saraswat, Frank D. Valencia, and Bjorn Victor. Linearity and persistence in the pi-calculus. unpublished, 2004.

[PV01] Catuscia Palamidessi and Frank Valencia. A temporal concurrent constraint programming calculus. In Toby Walsh, editor, *Proc. of the 7th International Conference on Principles and Practice of Constraint Programming*, volume 2239, pages 302–316. LNCS, Springer-Verlag, 2001.

[QvRDC06] Luis Quesada, Peter van Roy, Yves Deville, and Raphael Collet. Using dominators for solving constrained path problems. In *PADL'06*, 2006.

[Rip01] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network, 2001.

[Ros94] A. W. Roscoe. Model-checking csp. *A classical mind: essays in honour of C. A. R. Hoare*, pages 353–378, 1994. 0-13-294844-3.

[RP91] V. Saraswat, M. Rinard and P. Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91*, pages 333–352, jan 1991.

[RPS$^+$04] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Y. Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.

[RR05] J. Rohrer and M. Roth. Mute: Simple, anonymous file sharing, 2005. Available at `http://mute-net.sourceforge.net/howAnts.shtml`.

[RS98] P. Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol. a cautionary tale. *Inf. Process. Lett.*, 65(1):7–10, 1998.

[RSG$^+$01] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols.* Addison-Wesley, 2001.

[RSS01] Aviv Regev, William Silverman, and Ehud Y. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pacific Symposium on Biocomputing*, pages 459–470, 2001.

[Rud00] Carsten Rudolph. Considering non-malleability in formal models for cryptographic protocols. In *Proceedings of Workshop on Issues in the Theory of Security (WITS'00)*, 2000.

[Rue86] Rainer A. Rueppel. *Analysis and design of stream ciphers.* Springer-Verlag New York, Inc., New York, NY, USA, 1986.

[Sch96a] S. Schneider. Using csp for protocol analysis: the needham-schroeder public key protocol. Technical report, Royal Holloway, 1996.

[Sch96b] Steve Schneider. Modelling security properties with CSP. Technical Report CSD-TR-96-04, Department of Computer Science, Egham, Surrey TW20 0EX, England, 1996.

[Sch96c] Steve Schneider. Security properties and csp. In *SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 174, Washington, DC, USA, 1996. IEEE Computer Society.

[SK86] Abraham Silberschatz and Henry F. Korth. *Database System Concepts, 1st Edition.* McGraw-Hill Book Company, 1986.

[SL03] Aameek Singh and Ling Liu. Trustme: Anonymous management of trust relationships in decentralized p2p systems. In *IEEE Intl. Conf. on Peer-to-Peer Computing 2003.* IEEE press, September 2003.

[SL04] Mudhakar Srivatsa and Ling Liu. Vulnerabilities and security threats in structured peer-to-peer systems: A quantitative analysis. In *20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 252–261, 2004. Available at http://citeseer.csail.mit.edu/656519.html.

[SPG91] A. Silberschatz, J. Peterson, and P. Galvin. *Operating System Concepts.* 3 edition, 1991.

[SRP91] V. Saraswat, M. Rinard, and P. Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91*, pages 333–352, Jan 1991.

[SW01] Davide Sangiorgi and David Walker. *PI-Calculus: A Theory of Mobile Processes.* Cambridge University Press, New York, NY, USA, 2001.

[WHY$^+$04] Helen J. Wang, Yih-Chun Hu, Chun Yuan, Zheng Zhang, and Yi-Min Wang. Friends troubleshooting network: Towards privacy-preserving, automatic troubleshooting. In Geoffrey M. Voelker and Scott Shenker, editors, *IPTPS*, volume 3279 of *Lecture Notes in Computer Science*, pages 184–194. Springer, 2004.

[ZS00] Yongguang Zhang and Bikramjit Singh. A Multi-Layer IPsec protocol. In *USENIX 2000*, pages 213–228, 2000.

Appendices

A An introduction to Petri Nets

Petri nets are an abstract formal model used to describe concurrent an asynchronous systems. In this model it is possible to verify properties of a system, as constraints that can never be broken. It basic model consists of a directed graph where two kind of nodes are available: places and transitions. Places represents states of a process and transitions the synchronisation methods between states. This model is well suited to represent sequential and static behaviour of processes, as well as the dynamic properties and the execution of concurrent processes. We refer the reader to [Pet77] for deeper description of the model.

A.1 Multisets

A multiset is a set where the multiplicities of its elements matters.
Multisets could have infinite multiplicities. This is represented by including an extra element ∞ to the natural numbers. Multisets support addition $+$ and multiset inclusion $\leq$.

A.2 General Petri nets

A general Petri net is a place transition system consisting of a set of conditions P, a set of events T and a set of arcs connecting both of them. There are two types of arcs, the precondition map *pre*, which to each $t \in T$ assigns a multiset $pre(t)$ (traditionally written ${}^{\cdot}t$) over P and a postcondition map *post* which to each $t \in T$ assigns a ∞-multiset $post(t)$ ($t^{\cdot}$) over P. Petri nets also include a Capacity function Cap, an ∞-multiset over P, which assigns to each condition its respective multiplicity.

Token game for general nets.- A marking is a very important concept in Petri nets, since it captures the notion of a distributed global state. A marking is represented by the presence of tokens on a condition. The number of tokens denotes the multiplicity of each condition.

Markings can change as events occur, moving tokens from the event preconditions to its postconditions by what is called the token game of nets. For M, M' markings and $t \in T$ we define

$$M \xrightarrow{t} M' \text{ iff } {}^{\cdot}t \leq M \wedge M' = M -^{\cdot} t + t^{\cdot}$$

An event t is said to have concession at a marking M iff its occurrence leads to a marking.

A.3 Basic Nets

Basic nets are just a instantiation of a general Petri net, where in all the multisets the multiplicities are either 0 or 1, and so can be regarded as sets. In this case, the capacity function assigns 1 to every condition in such a way that markings become just simply subsets of conditions.

A basic Petri net consists of a set of conditions B, a set of events E and two maps. A *precondition* map $pre : E \to Pow(B)$, and a *postcondition* map $post : E \to Pow(B)$.

We can denote ${}^{\cdot}e$ for the preconditions and $e^{\cdot}$ for the postconditions of $e \in E$ requiring that ${}^{\cdot}e \cup e^{\cdot} \neq \emptyset$

Token game for basic nets.- For markings $M, M' \subseteq B$ and event $e \in E$, define

$$M \xrightarrow{e} M' \text{ iff}$$

(1) ${}^{\cdot}e \subseteq M$ & $(M \backslash {}^{\cdot}e) \cap e^{\cdot} = \emptyset$ and

(2) $M' = (M \backslash {}^{\cdot}e) \cup e^{\cdot}$

A.4 Nets with persistent conditions

A net with persistent conditions is a modification of a basic net. It allows certain conditions to be persistent in such a way that any number of events can make use of them as preconditions which never cease to hold. This conditions can also act as postconditions for several events without generating any conflict.

Now, amongst the general conditions of the basic net, are the subset of persistent conditions P, forming in this way a persistent net.

The general net's capacity function will be either 1 or ∞ on a condition, being ∞ precisely on the persistent conditions. When p is persistent, $p \in e^{\cdot}$ is interpreted in the general net as arc weight $(e^{\cdot})_p = \infty$, and $p \in^{\cdot} e$ as $({}^{\cdot}e)_p = 1$.

Token game with persistent conditions.- The token game is modified to account for the subset of persistent conditions P. Let M and M' be markings (*i.e.* subsets of conditions), and e an event. Define

$$M \xrightarrow{e} M' \text{ iff}$$

(1) ${}^{\cdot}e \subseteq M$ & $(M \backslash ({}^{\cdot}e \cup P)) \cap e^{\cdot} = \emptyset$ and

(2) $M' = (M \backslash {}^{\cdot}e) \cup e^{\cdot} \cup (M \cap P)$.