Formally Reasoning About Security Issues in P2P Protocols: A Case Study

Andrés A. Aristizábal[1], Hugo A. López[1], Camilo Rueda[1], and Frank D. Valencia[2]

[1] AVISPA Research Group. Pontificia Universidad Javeriana Cali, Colombia
aaaristizabal@puj.edu.co, {halopez,crueda}@cic.puj.edu.co
[2] CNRS-LIX École Polytechnique de Paris, France.
frank.valencia@lix.polytechnique.fr

Abstract. Peer-to-Peer (P2P) systems can be seen as highly dynamic distributed systems designed for very specific purposes, such as resources sharing in collaborative settings. Because of their ubiquity, it is fundamental to provide techniques for formally proving properties of the communication protocols underlying those systems. In this paper we present a formal specification of MUTE, a protocol for P2P systems, modelled in the SPL process calculus. Furthermore, we use the SPL reasoning techniques to show how the protocol enjoys a secrecy property against outsider attacks. By formally modeling and analyzing a real-world, yet informally specified protocol, we bear witness to the applicability of SPL as a formalism to specify security protocols as well as the flexibility of its reasoning techniques. This paper represents our first approach towards the use of process calculi, in particular SPL, for the specification and reasoning of P2P protocols.

1 Introduction

Peer-to-Peer (P2P) protocols are widely used for communication in distributed systems, providing an accurate and efficient way to perform certain important tasks, including information retrieval and routing. Protocols for P2P systems are then 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 nowadays computing world because of its huge amount of benefits, such as its architecture cost, scalability, viability, and resource aggregation of distributed management infrastructures. Essentially this kind of systems are used to obtain the major benefit from distributed resources to perform a function in a real decentralised manner. In this way, these systems are scalable since they avoid dependencies on centralised nodes, and they also have a low cost infrastructure, since they enable direct communication between the participants of these systems.

The P2P protocols used in various tools have to maintain a certain amount of important properties to guarantee their well functioning. One class of the most relevant 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 [1]. 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 protocols, rigorous analysis about security matters are still open to development.

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 [2,3,4,5,6,7]: 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 [8,9,10,11,12,13]. 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 process calculi for security include π and the Spi calculus [4,14], the CSP process algebra [6], and more recently, the Secure Protocol Language (SPL) by Crazzolara and Winskel [15].

Despite of the recent interest in P2P systems and applications, little work has been done in the formal analysis of P2P protocols. Previous works related with process calculi include the work within the Pepito project [16] in the verification of correctness properties in static versions of P2P protocols using CCS variants [17]. Other analyses has been made for specific P2P functionalities, like [18] and [19]. To the best of our knowledge, this paper thus constitutes the first work concerned to the modelling of P2P protocols using a formal approach such as process calculi. We illustrate our approach by formalizing MUTE, a P2P tool for sharing and transmitting resources in a highly dynamic distributed network [20]. MUTE is based on its particular searching protocol, which claims to guarantee an anonymous and secure way of communicating data through the entire P2P network. In spite of being a well-known P2P protocol, MUTE has only been informally described. We shall use SPL to give a formal specification of MUTE for the first time. Based on such a specification, we use SPL reasoning techniques to verify a secrecy property for MUTE, namely, that a malicious, outsider agent can not access messages within MUTE network.

The main contribution of this paper is a formal model of MUTE which abstracts away from details not concerned with security issues. By means of this model and its associated security proofs, we bear witness of the applicability of SPL and its proof techniques for modelling and reasoning about protocols. The work in the present paper represents our first approach towards the use of SPL as formalism for specifying and verifying P2P protocols.

The paper is structured as follows. Next we present a brief summary of preliminaries, including a short introduction to the SPL calculus. In section 3, we explain the MUTE protocol, presenting an intuitive representation, as well as its formalisation on SPL. In section 4, we use SPL proof techniques to verify the secrecy property for messages behind an outsider attacker in the MUTE protocol. In section 5 we discuss some related work and in the last chapter we give out some concluding remarks, as well as future work.

2 Preliminaries

This section presents a brief overview of SPL (Security Protocol language), a process calculus for security protocols proposed by Winskel and Crazzolara in [15]. A full coverage of the calculus is given in [21].

2.1 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 [22], which states that cryptography is unbreakable and the spy is an active intruder capable of intercept, modify, replay and suppress messages. The calculus is operationally defined in terms of configurations containing items of information (messages) which can only increase during evolution, modelling the fact that, in an open network an intruder can see and remember any message that was ever in transit.

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 [23]) 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 $\boldsymbol{x}, \boldsymbol{\chi}$ and $\boldsymbol{\psi}$ 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	$k ::= Pub(v) \mid Priv(v) \mid Key(\boldsymbol{v}) \mid \chi, \chi', ...$
Messages	$M, M' ::= v \mid k \mid (M, M') \mid \{M\}_k \mid \psi, \psi', ...$
Processes	$p ::= out\, new(\boldsymbol{x})\, M.p \mid in\, pat \boldsymbol{x}\boldsymbol{\chi}\boldsymbol{\psi}\, M.p \mid \|_{i \in I}\, P_i \mid !P$

(a) SPL Syntax

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

(b) SPL Transition Semantics

The rest of the elements of SPL syntactic set are defined in Table 1(a), where $Pub(v)$, $Priv(v)$ and $Key(\boldsymbol{v})$ denote the generation of public, private and shared keys respectively. We use the vector notation $\boldsymbol{s}$ to denote a list of elements, possibly empty, $s_1, s_2, \ldots, s_n$.

2.2 Intuitive Description and Conventions

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

The output process $out\, new(\boldsymbol{x})\, M.p$ generates a set of fresh distinct names (nonces) $\boldsymbol{n} = n_1, n_2, \ldots, n_m$ for the variables $\boldsymbol{x} = x_1, x_2, \ldots, x_m$. Then it outputs the message $M[\boldsymbol{n}/\boldsymbol{x}]$ (i.e., M with each x_i replaced with n_i) in the store and resumes as the process $p[\boldsymbol{n}/\boldsymbol{x}]$. The output process binds the occurrence of the variables $\boldsymbol{x}$ in M and p. As an example of a typical output, $p_A = out\, new(\boldsymbol{x})\, \{x, A\}_{Pub(B)}.p$ 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 $out\, new(\boldsymbol{x})\, M.p$ simply as $out\, M.p$ if the $\boldsymbol{x}$ is empty.

The input process $in\, pat\, \boldsymbol{x}\boldsymbol{\chi}\boldsymbol{\psi}\, M.p$ waits for a message in the store that matches (the pattern) the message M for some instantiation of its variables $\boldsymbol{x}$, $\boldsymbol{\chi}$ and $\boldsymbol{\psi}$. The process resumes as p with the chosen instantiation. The input process $in\, pat\, \boldsymbol{x}\boldsymbol{\chi}\boldsymbol{\psi}\, M.p$ is the other binder in SPL binding the occurrences of $\boldsymbol{x}\boldsymbol{\chi}\boldsymbol{\psi}$ in M and 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 n for x and A for Z. When no confusion arises we will sometimes abbreviate $in\, pat\, \boldsymbol{x}\boldsymbol{\chi}\boldsymbol{\psi}\, M.p$ 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 $P_1 \parallel P_2 \parallel \ldots \parallel P_n$ to mean $\|_{i \in \{1,2,\ldots n\}}\ P_i$.

The syntactic notions of free variables and closed process/message are defined in the obvious 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.

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 so-far generated), 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 $\alpha :: out\, new(\boldsymbol{n})\, M \mid in\, M \mid i : \alpha$. where $\boldsymbol{n}$ 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 1(b) define the transitions between configurations. The rules are easily seen to realize the intuitive behaviour 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.

Event-Based Semantics Although transition semantics provide an appropriate method to show the behaviour 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* [21] 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 [21], 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 1 gives the general form of an SPL event. In the Appendix we will give the events for the protocol MUTE according to the SPL Event-Semantics. The exact definition of each element of the semantics can be can be found in [21]. For space limitations, here we shall recall some and illustrate others.

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

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

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

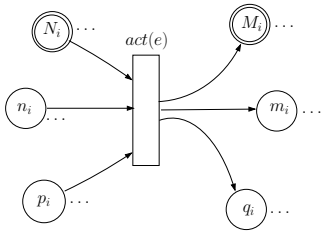


Fig. 1. Events and transitions of SPL event based semantics. p_i and q_i denote control conditions, n_i and m_i name conditions and N_i, M_i output conditions. Double circled conditions denote persistent events.

Relating Transition and Event Based Semantics Transition and event based semantics are strongly related in SPL by the following theorem from [21]. 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.

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:

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

Where $\boldsymbol{n}$ are distinct names

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

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

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

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

3 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 [20]. It is based on a particular searching protocol, which claims to guarantee an anonymous and secure way of communicating data through the entire P2P network. In spite of being a real life protocol, MUTE has only been informally described. Following an original approach, we shall use SPL to give a formal specification of the MUTE protocol.

This protocol aims to provide an easy and effective search while protecting the privacy of the participants involved. It is inspired in the behaviour of ants in the search for food. The analogy is accomplished 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, like the ants that do not know 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[3] .

Since MUTE 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 only knows the existence of other peers in the network by means of direct connections between them, using them to accomplish the search and transmission of data. This nodes are called "neighbours" 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 an important property in the protocol , 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] Abstracting from the MUTE website, available at [20]

3.1 Abstraction of MUTE

In spite of being already implemented and used as a tool for downloading and file-sharing, 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, leaving aside elements in the protocol such as the dynamic nature of the nodes in the P2P network, as well as their insertion and deletion. These considerations, albeit important in P2P systems, can be modelled by including elements in the protocol guaranteeing the freshness of every request produced. Phases of key establishment, widely studied in several works [24,25,26], are also not considered in the model.

Definition 1 (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 $Files$, $Keywords(A)$ 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 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 2 (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 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 with the keyword Kw and a nonce N to all its neighbors in order to find a particular answer, Also suppose that B is the agent that has the desired answer which A is searching for, which is willing to send a response. In this case, B can be any node in G 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 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 initial sender A. Figure 2 give a representation of the above description.

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

Fig. 2. 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.

3.2 MUTE Specification on SPL

We use the core of the MUTE protocol in order to establish some security properties. The phases that we shall consider are the ones that involve the transmission of the keyword, the response message and the keys, leaving behind the phases of connection, and the submessages that include plaintext. We assume that the symmetric keys are equivalent (i.e. $key(A, B) = key(B, A)$). The formal model is presented in Figure 3 [4].

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

Fig. 3. MUTE specification on SPL

Intuitive description of the specification We assume that the topology of the net has already been established. The agent starts searching for an own keyword. This agent broadcasts the desired keyword to all its neighbours. Its neighbours receive the message and see if the keyword matches one of their files, if at least one of the neighbours have the requested keyword, it will broadcast a response message, such that eventually the one searching for the keyword will get it and understand it as an answer to its request. The message will be forwarded by all the agents until it reaches its destination. Otherwise, if the keyword does not match any file of the agent, then it will broadcast it to its neighbours asking them for the same keyword. The choice of having or not the right file is modeled in a non-deterministic way. This model abstracts way from issues such as the search for the best path, since it has no impact in secrecy.

3.3 MUTE Secrecy Proofs for a Outsider Spy

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

Definition of the Spy Using a well studied model of spy [21], a possible attacker over the network is presented in table 1

[4] Notation $(\|_{i \in I}\ P_i)\,.\,R$, representing the execution of process R once parallel composition $\|_{i \in I}\ P_i$ is fully executed, can be easily encoded using elements in the language. See [27] for more details.

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	$Spy_7 \equiv in\,Priv(x).in\,\psi.out\,\{\psi\}_{Priv(x)}$
Verify signatures with available keys	$Spy_8 \equiv in\,x.in\,\{\psi\}_{Priv(x)}.out\,\psi$
Create new random values	$Spy_9 \equiv out\,new(\boldsymbol{n})\boldsymbol{n}$

Table 1. Spy model for the MUTE protocol

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

$$Spy \equiv \|_{i\in\{1\ldots9\}} Spy_i \tag{1}$$

In this way, the complete protocol includes the specification of MUTE, $SecureMute$ in Figure 3, in parallel with the Spy:

$$MUTE \equiv SecureMUTE \| !Spy \tag{2}$$

Let us recall some elements. 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 $Files(A)$ (See section 3.1).

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

Definition 3 (Run of a Protocol). *A run of a process $p = p_0$ is a sequence*

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

We shall use in the theorems a binary relation $\sqsubset$ between messages. Intuitively $M \sqsubset M'$ means message M is a subexpression of message M'. (See Appendix B for the exact definition)

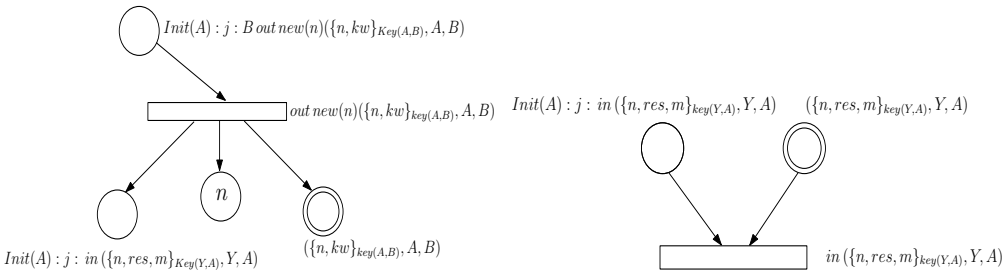
The Events of MUTE MUTE has three kind of events, according to each role of the agents in the network:

Definition 4 (Events in MUTE). *The event e_w is an event in the set*

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

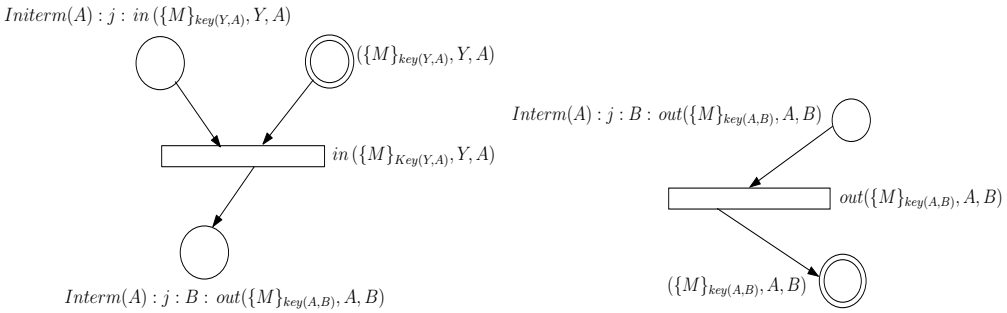
Where the events are graphically represented in figures 4, 5 and 6.

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 4(b)), 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 4(a).



(a) Initiator Output action (b) Initiator Input action

Fig. 4. Initiator Events



(a) Input action (b) Output action

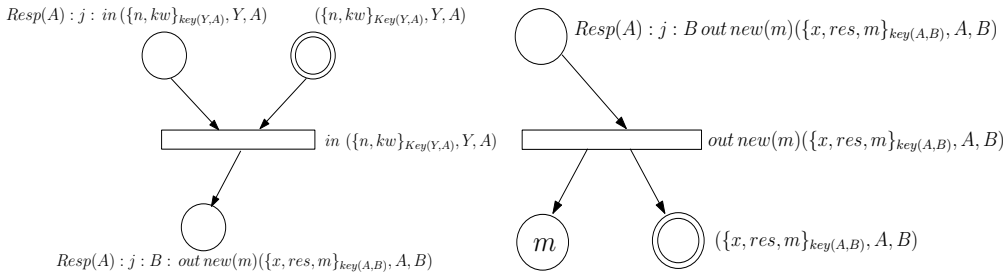
Fig. 5. Intermediator Events

Intermediator Events: Each agent acting as an intermediator has to forward the received messages. The figure 5(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 5(b)) completes the intermeditator behavior, forwarding received messages M to one of the neighbors by means of an output $out\,(\{M\}_{Key(A,B)}, A, B)$.

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 6(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 6(b)).

3.4 Secrecy Properties

The first secrecy theorem for the MUTE protocol concerns the shared keys of neighbours. If these are not corrupted from the beginning of the protocol, and the peers behave as the protocol states then the keys will not be leaked during a protocol run. If we assume that $key(X, Y) \not\sqsubseteq t_0$,



(a) Input action (b) Output action

Fig. 6. Responder Events

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* $\langle MUTE, s_0, t_0\rangle \xrightarrow{e_1} \cdots \xrightarrow{e_v} \langle p_v, s_v, t_v\rangle \xrightarrow{e_{v+1}} \ldots$ *and* $A_0, B_0 \in Peers(G)$*, if* $key(A_0, B_0) \not\sqsubseteq t_0$ *then for each* $w \geq 0$ *in the run* $key(A_0, B_0) \not\sqsubseteq t_w$

Proof. Outline Following the proof technique given in [21] the proof proceeds by stating a property associated with shared keys not appearing as a cleartext in the protocol. Then we assume a run which contains an event which violates the property stated before, and using dependencies among events within the protocol, we derive a contradiction. (The complete proof can be found in Appendix D1). By proving that shared keys never appear in the cleartext during a run of the protocol, we can guarantee that a Spy outside the P2P network cannot have access to them. Later on we will see the importance of this property for ensuring security in the protocol.

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 peer to peer group.

Theorem 3. *Given a run of MUTE* $\langle MUTE, s_0, t_0\rangle \xrightarrow{e_1} \cdots \xrightarrow{e_v} \langle p_v, s_v, t_v\rangle \xrightarrow{e_{v+1}} \ldots$, $A_0 \in Peers(G)$ *and* $kw_0 \in Keywords(A_0)$*, if for all A,B* $\in Peers(G)$*,* $key(A, B) \not\sqsubseteq t_0$*, where* $B \in ngb(A)$ *and the run contains the 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 an index,* B_0 *is an index which belongs to the set* $ngb(A_0)$*, and* n_0 *is a name, then for every* $w \geq 0$ *in the run* $kw_0 \notin t_w$

Proof. Outline Following the proof technique given in [21] the proof proceeds by stating that the shared keys are never leaked during a run of the protocol (Theorem 2). We state a stronger property Q which holds for all keywords not appearing as a cleartext during a run of the protocol. Then we assume an event which violates property Q, and using dependencies among events within the protocol we derive a contradiction. (The complete proof can be found in Appendix D2).

By proving that the keyword sent by the initiator peer as a request never appears in the cleartext during a run of the protocol, we can affirm that a Spy outside of the network will never know that keyword, so he will never recognise the file a sender is requesting.

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* $\langle MUTE, s_0, t_0\rangle \xrightarrow{e_1} \cdots \xrightarrow{e_v} \langle p_v, s_v, t_v\rangle \xrightarrow{e_{v+1}} \ldots$ *and* $A_0 \in Peers(G)$ *and* $res_0 \in Headers(B_0)$*, if for all A,B* $\in Peers(G)$*,* $key(A,B) \not\sqsubseteq t_0$*, 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 an index, B_0 is an index which belongs to the set $ngb(A_0)$ and n_0, m_0 are names, then for every $w \geq 0$ $res_0 \notin t_w$

Proof. Outline The proof is analogous to that of Theorem 2 (The complete proof can be found in Appendix D3).

If we prove that the answer header sent by the receiver, never appears in the cleartext during a run of the protocol we can manage to guarantee that a Spy outside the peer to peer network will never know or access the file.

4 Concluding Remarks and Future Work

The use of process calculi allows us to formalise communication protocols leaving aside technical details, transforming complex distributed algorithms into abstract models syntactically close to their descriptions in pseudo-code. In particular, the use of SPL calculus lets us model several processes involved in the protocol without losing dependencies among them, in order to verify security properties along all the runs of a given protocol. In this way, these properties essential to communication (P2P) protocols can be easily verified. We demonstrate the above by giving the first formal description MUTE and by showing secrecy properties for messages w.r.t. a outsider attacker in the MUTE protocol. This bears witness of the specification power of SPL and its reasoning techniques.

We have proved the secrecy property for an outsider in the MUTE protocol. However, it is crucial to explore security properties for threats inside the P2P network or an outsider who can masquerade as a trusted peer. Our future work will study secrecy w.r.t. to a malicious insider in the MUTE protocol. We shall also explore the SPL expressiveness in order to model new cutting-edge protocols, using its own reasoning techniques, or extending it in order to verify other important security properties like non-traceability and non-malleability.

References

1. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.
2. Andrés A. Aristizábal and Hugo A. López. Using process calculi to model and verify security properties in real life communication protocols. Bsc. thesis, Pontificia Universidad Javeriana, January 2006.
3. 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.

4. Johannes Borgstrôm, Uwe Nestmann, Luc Onana Alima, and Dilian Gurov. Verifying a structured peer-to-peer overlay network: The static case. In Paola Quaglia Corrado Priami, editor, *Global Computing: IST/FET International Workshop, GC 2004 Rovereto, Italy, March 9-12, 2004 Revised Selected Papers*, volume 3267 of *Lecture Notes in Computer Science*, page 250. Springer, 2004.
5. 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.
6. 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.
7. 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.
8. Federico Crazzolara. Language, semantics, and methods for security protocols. Doctoral Dissertation DS-03-4, brics, daimi, May 2003. PhD thesis. xii+160.
9. Federico Crazzolara and Glynn Winskel. Events in security protocols. In *ACM Conference on Computer and Communications Security*, pages 96–105, 2001.
10. 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.
11. Paul B. Garrett. *Making, Breaking Codes: Introduction to Cryptology*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
12. Julian Gutiérrez, Jorge Andrés Pérez, Camilo Rueda, and Frank Valencia. A Timed Process Calculus for Modeling and Verifying Biological Systems. Submitted for Publication, January 2006.
13. Seif Haridi and Thom Sjôland. Pepito - peer-to-peer: Implementation and theory, 2002. Project Proposal.
14. C. A. R. Hoare. Communicating Sequential Processes. *Commun. ACM*, 26(1):100–106, 1983.
15. J. Krivine and V. Danos. Formal molecular biology done in CCS-R. In *BioConcur 2003, Workshop on Concurrent Models in Molecular Biology*, 2003.
16. Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.
17. Giuseppe Milicia. χ-Spaces: Programming Security Protocols. In *Proceedings of the 14th Nordic Workshop on Programming Theory (NWPT'02)*, November 2002.
18. Robin Milner. *Communication and concurrency*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1995.
19. Robin Milner. *Communicating and Mobile systems. The Pi Calculus*. Cambridge University Press, 1999.
20. 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.
21. Charles Perkins. IP Mobility Support - RFC2002. IETF RFC Publication, 1996.
22. James L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, 1977.
23. G. D. Plotkin. A structural approach to operational semantics. Technical report, University of Aarhus, 1981.
24. 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.
25. 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.
26. V. Saraswat, M. Rinard and P. Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91*, pages 333–352, jan 1991.
27. J. Rohrer and M. Roth. Mute: Simple, anonymous file sharing, 2005. Available at `http://mute-net.sourceforge.net/howAnts.shtml`.
28. Mudhakar Srivatsa and Ling Liu. Vulnerabilities and security threats in structured peer-to-peer systems: A quantitative analysis.

29. 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.

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 [28] 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 an 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) $M' = (M \backslash {}^{\cdot}e) \cup e^{\cdot} \cup (M \cap P)$.

B General Proof principles [21]

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:

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

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.
Well-foundedness.- *Given a property P on configurations, if a run $\langle p_0, s_0, t_0\rangle \xrightarrow{e_1} \ldots \xrightarrow{e_r} \langle p_r, s_r, t_r\rangle \xrightarrow{e_{r+1}} \ldots$, contains configurations s.t. $P(p_0, s_0, t_0)$ and $\neg P(p_j, s_j, t_j)$, then there is an event e_h, $0 < h \leq j$, s.t. $P(p_i, s_i, t_i)$ 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 $Fresh(m, e)$
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:

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

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, $j < i$, s.t. $b \in e_j^o$.
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, $j < i$, s.t. $M \in e_j^o$
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
Message Surroundings.- *Given 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:

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

C Secrecy Property (Full Proofs)

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 5. *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 4) 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 3.3, 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.

C.1 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 6. *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 4) 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

$$\sigma(kw_0, e_v^o - e_{v-1}^o) \not\subseteq \{(\{n_0, kw_0\}_{key(A_0,B_0)}, A_0, B_0)\} \tag{3}$$

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 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 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.*

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

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

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

where $A \in Peers(G)$ 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 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 is reached, because from the output principle if follows that $e_v^o - e_{v-1}^o = \{\{n_0, kw_0\}_{key(A,B)}, A, B\}$. Then, from the definition of message surroundings and Property 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: $(M = (n, res, m))$

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

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 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 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 $\sigma(kw_0, t_{u-1}) \not\subseteq (\{n_0, kw_0\}_{key(A_0,B_0)}, A_0, B_0)$ 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 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 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 5). Since this there is no possible way for a spy to reach kw_0, e_v is not a spy event.

C.2 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 7. *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 4) 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.

$$\sigma(res_0, e_v^o - e_{v-1}^o) \not\subseteq \{(\{n_0, res_0, m_0\}_{key(A_0,B_0)}, A_0, B_0)\} \tag{4}$$

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 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 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 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 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 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 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 5). Since this there is no possible way for a spy to reach kw_0, e_v is not a spy event.