Using a Declarative Process Language for P2P Protocols

Andrés A. Aristizábal
Universidad Javeriana Cali, Colombia
aaaristizabal@puj.edu.co

Hugo A. López
Universidad Javeriana Cali, Colombia
halopez@puj.edu.co

Camilo Rueda
Universidad Javeriana Cali, Colombia
crueda@atlas.puj.edu.co

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 model of MUTE, a protocol for P2P systems, in the SPL; a specification language with a striking resemblance to Concurrent Constraint Programming. Furthermore, we use the SPL reasoning techniques to show the protocol enjoys a secrecy property againts outsider attacks. By formally modeling and analyzing a popular (albeit never specified) protocol, we bear witness to the applicability of SPL as a formalism to model and reason about security protocols as well as flexibility of the its reasoning techniques.

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 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. 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 points, 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 its well functioning. One class of the most relevant P2P protocols are those concerned to security. Properties like secrecy, anonymity and non-traceability have been studied in the literature in order to overcome security risks [9]. Anonymity itself, is considered of the essence of any peer to peer protocol, since the participants on the network wish to establish communications and share resources without revealing their identity. Similarly, secrecy is considered crucial, since messages transmitted and managed between the distributed components in the network shall be kept as a secret for an entity outside the peer to peer group.

Despite the popularity of this kind of protocols, 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 modelling P2P protocols using process languages.

MUTE is a P2P tool for sharing and transmitting resources in a highly dynamic distributed network [12]. It is based on its particular searching protocol, which claims to guarantee an anonymous way of communicating data, in a secure way, through all the P2P network. In spite of being a very well known peer to peer protocol, MUTE has only been informally described.

We shall use SPL, a specification language for security protocols developed by Winskel and Crazzolara to give MUTE a formal specification for the first time. We then use SPL reasoning techniques to verify a secrecy property againts attacks of an outsider.

SPL has strong a similarity with Concurrent Constraint Programming (CCP) [13]. Just like CCP, SPL is operationally defined in terms of configurations containing items of information (messages) which *can only increase* during evolution. Such a monotonic evolution of information is akin to the notion of *monotonic store* central to CCP and a source of its simplicity.

One contribution of this paper is to give, to our knowledge, the first formal model of MUTE which abstracts away from details not concerned with secrecy issues. Another contribution is to bear witness of the applicability of SPL and its proof techniques for modelling and reasoning about protocols. The work in the present paper represent our first approach towards the use of SPL as formalism for specifying and verifying P2P protocols. The current work has been submitted to a workshop in P2P systems.

We shall proceed as follows: 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 its 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, taking into account an outsider malicious entity.

The paper is structured as follows. In the next section we present a brief summary of preliminaries, including a short introduction of the SPL calculus. In section 3, we explain the MUTE protocol, presenting an intuitive representation, as well as its formalisation on SPL. In the following section, we follow the SPL proof techniques scheme to verify the secrecy property for messages behind a passive outsider 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 [3]. The full coverage of the calculus is given in [2].

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 [4], 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.

2.1.1 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, uniques from previous choices [10]) 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 $P, Q, R, ...$

The rest of the elements of SPL syntactic set are defined in Table 1(a), 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, $s_1, s_2, \ldots, s_n$.

2.2 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 $\vec{x}$ is empty.

The input process $in\, pat\, \vec{x}\vec{\chi}\vec{\psi}\, M.p$ waits for a message in the store that matches (the pattern) the message M for some instantiation of its variables $\vec{x}$, $\vec{\chi}$ and $\vec{\psi}$. The process resumes as p with the chosen instantiation. 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 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 $in\, pat\, \vec{x}\vec{\chi}\vec{\psi}\, M.p$ 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

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	$p ::= out\, new(\vec{x})\, M.p$ $\mid in\, pat\vec{x}\vec{\chi}\vec{\psi}\, M.p$ $\mid \ \Vert_{i\in I}\ P_i$ $\mid !P$

(a) 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 $p'_i = p_i$

(b) SPL Transition Semantics

P in parallel. We sometimes write $P_1 \parallel P_2 \parallel \ldots \parallel P_n$ to mean $\Vert_{i\in\{1,2,\ldots n\}}\ P_i$.

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.

2.2.1 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 $\alpha ::= out\, new(\vec{n})\, M \mid in\, M \mid i : \alpha$. 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 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.

2.2.2 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* [2] 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 [2], 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

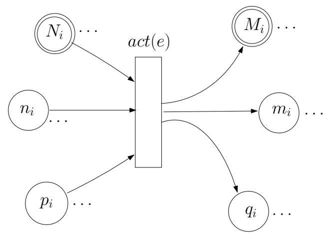


Figure 1. Events and transitions of SPL event based semantics. p_i and q_i denote control conditions, n_i and m_i name conditions and N_i, M_i output conditions. Double circled conditions denote persistent events.

$e^{\cdot} = \{e^c, e^o, e^n\}$ 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 [2]. For space limitations,here we shall recall some and illustrate others.

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

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

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 $Ic(\|_{i\in I}\ P_i) = \bigcup_{i\in I}\{i : c \mid c \in Ic(P_i)\}$

2.2.3 Relating Transition and Event Based Semantics

Transition and event based semantics are strongly related in SPL by the following theorem from [2]. 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 2.1. *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.

2.2.4 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:

$$
\begin{array}{lll}
Ev(out\,new\,\vec{x}M.p) & = & \{\,\mathbf{Out}\,(out\,new\,\vec{x}M.p;\ \vec{n})\} \\
 & & \cup \bigcup\{Ev(p[\vec{n}/\vec{x}])\} \\
\multicolumn{3}{l}{\text{Where } \vec{n} \text{ are distinct names}} \\
Ev(in\,pat\,\vec{x}\vec{\chi}\vec{\psi}M.p) & = & \{\mathbf{In}(in\,pat\,\vec{x}\vec{\chi}\vec{\psi}M.p;\ \vec{n}, \vec{k}, \vec{L})\} \cup \\
 & & \bigcup\{Ev(p[\vec{n}/\vec{x}, \vec{k}/\vec{\chi}, \vec{L}/\vec{\psi}])\} \\
\multicolumn{3}{l}{\text{Where } \vec{n} \text{ names, } \vec{k} \text{ are keys, and } \vec{L} \text{ are closed messages}} \\
Ev(\|_{i\in I} p_i) & = & \bigcup_{i\in I} i : Ev(p_i)
\end{array}
$$

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 [12]. This protocol is composed of two main phases, the searching and the routing part. We will focus directly in its first phase, since it is the most related to the security concerns proposed in 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 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.[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 "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 one 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.

[1] Abstracting from the MUTE website, available at [12]

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

Figure 2. Dolev Yao Model of the MUTE **protocol**

3.1 Dolev Yao Representation for the MUTE protocol

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.

We shall represent a P2P network as an undirected graph G whose nodes represent the peers and whose edges represent the direct connections among peers. 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 neighbours of X. We use the Dolev Yoe notation $X \longrightarrow Y : M$ stating that X sends a message M to Y. For Example, consider a P2P network G with peers A, B. Suppose that A is the initiator of the protocol and B is the responder. In this case, B can be any node inside the network, with the desired file on its store. So, A requests a particular file he wishes to download. For this purpose he sends the request to the network, by broadcasting it to his neighbours. This request includes a keyword $kw \in Keywords$ which will match the desired file. Along the searching path an unknown amount of peers will forward the request until it reaches B, a peer which has the correct file, st $\exists f \in Files(B)$ and $kw \in f$, where $Files$ means the set of all files in the network, $Files(A)$ means the set of files of the Agent or peer A in the network, $Keywords$ the set of keywords associated to the files $Files$, and $Keywords(A)$ the keywords associated to the peer A. Then, B sends its response by means of the header of the file, again by means of a broadcast through a series of forward steps, until it reaches the actual sender A. Figure 2 give a representation of the above description using Dolev Yao notation [4].

Here X, Y are variables which represent the forwarder peers along the path that goes from the Initiator to the responder node. This intermediate process may continue, until the target is reached. In the same way, these two variables will represent the peers which will forward the answer from the responder to the initiator. This process may be repeated several times as well.

Figure 3 illustrates a particular P2P network example in which A is the initiator, F_1, F_2, F_3 are the peers involved in the forwarding process and B is the receptor that sends its answer via the same three forwarders.

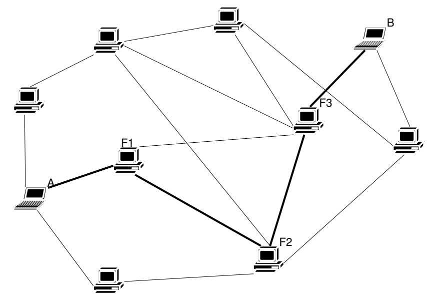


Figure 3. A typical MUTE **network topology**

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 $key(A, B) = key(B, A)$. The formal model in Figure 4.

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

Figure 4. MUTE **specification on SPL**

3.2.1 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 destiny. 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 Passive Outsider

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

3.4 Definition of the Spy

We use the definition of a powerful spy used in SPL [2] to model the ways of intrusion and attack that an agent can do.

1. Compose different messages into a single tuple
   $Spy_1 \equiv in\psi_1.in\,\psi_2.out\,\psi_1, \psi_2$
2. Decompose a compose message into more components
   $Spy_2 \equiv in\,\psi_1, \psi_2.out\,\psi_1.out\,\psi_2$
3. Encrypt any message with the keys that are available
   $Spy_3 \equiv in\,x.in\,\psi.out\,\{\psi\}_{Pub(x)}$
   $Spy_4 \equiv in\,Key(x,y).in\,\psi.out\,\{\psi\}_{Key(x,y)}$
4. Decrypt messages with available keys
   $Spy_5 \equiv in\,Priv(x).in\,\{\psi\}_{Pub(x)}.out\,\psi$
   $Spy_6 \equiv in\,Key(x,y).in\,\{\psi\}_{Key(x,y)}.out\,\psi$
5. Sign with available keys
   $Spy_7 \equiv Priv(x).in\,\psi.out\,\{\psi\}_{Priv(x)}$
6. Verify signatures with available keys
   $Spy_8 \equiv in\,x.in\,\{\psi\}_{Priv(x)}.out\,\psi$
7. Create new random values
   $Spy_9 \equiv out\,new(\vec{n})\vec{n}$
8. The Complete Spy is a parallel composition of the Spy_i processes:
   $Spy \equiv \|_{i\in\{1...9\}} Spy_i$

Figure 5. SPL spy model

In this way, the complete protocol includes the specification of MUTE, $SecureMute$ in Figure 4, in parallel with the Spy:

$$MUTE \equiv SecureMUTE \| !Spy \qquad (1)$$

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 3.1 (Run of a Protcol). *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)

3.5 Secrecy Properties

In the following theorems we shall refer to the events of MUTE which for the sake of space are explicitely given in Appendix C.

The first secrecy theorem for the MUTE protocol concerns the shared keys of neighbours. 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 3.2. *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 [2] 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 3.3. *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 [2] the proof proceeds by stating that the shared keys are never leaked during a run of the protocol (Theorem 3.2). 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 3.4. *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 3.2 (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 Related Work

To our knowledge, little work has been done in formalisation of P2P protocols using Process Calculi. In particular, the project Pepito [5] has started efforts in verification of properties using CCS variants in static versions of P2P protocols [1], in particular, correctness properties. Other analysis has been made for specific P2P functionalities, like [14] and [15]. However, to our knowledge, this is the first formal attempt using process calculi to model and reason about security properties in P2P protocols.

5 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 pseudocode. 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 protocol path. 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 wrt a passive outsider in the MUTE protocol. This bears witness of the specification power of SPL and its reasoning techniques.

We have proved the secrecy property for a passive outsider in the MUTE protocol. It will be interesting to explore security properties for threats inside the P2P network or an outsider who can masquerade as a trusted peer. Our future work will be the verification of the secrecy property for an insider in the MUTE protocol. In the same way, we shall 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] J. Borgström, U. Nestmann, L. O. Alima, and D. Gurov. Verifying a structured peer-to-peer overlay network: The static case. In TBD, editor, *Proc. Global Computing 2004*, Lecture Notes in Computer Science, page TBD. Springer, 2004.

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

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

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

[5] S. Haridi and T. Sjôland. Pepito - peer-to-peer: Implementation and theory, 2002.

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

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

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

[9] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, March 2002.
[10] C. Perkins. IP Mobility Support - RFC2002. IETF RFC Publication, 1996.
[11] J. L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, 1977.
[12] J. Rohrer and M. Roth. Mute: Simple, anonymous file sharing, 2005. Available at http://mute-net.sourceforge.net/howAnts.shtml.
[13] V. Saraswat. *Concurrent Constraint Programming*. The MIT Press, Cambridge, MA, 1993.
[14] M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured peer-to-peer systems: A quantitative analysis.
[15] H. J. Wang, Y.-C. Hu, C. Yuan, Z. Zhang, and Y.-M. Wang. Friends troubleshooting network: Towards privacy-preserving, automatic troubleshooting. In G. M. Voelker and S. 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 [11] 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 $pre : E \rightarrow Pow(B)$, and a *postcondition* map $post : E \rightarrow Pow(B)$.

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 [2]

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:

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

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

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 st $Fresh(n, e_j)$.

ii) Given a name n there is at most one event e_i st $Fresh(n, e_i)$.

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, $j < i$, st $b \in e_j^o$.

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, $j < i$, st $M \in e_j^o$

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 $Key(A)$.

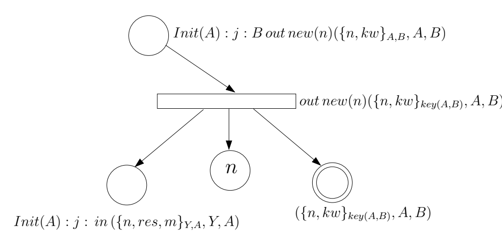
Let M and N be two messages. Define $\sigma(N, M)$ the surroundings of N in M inductively as follows:

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

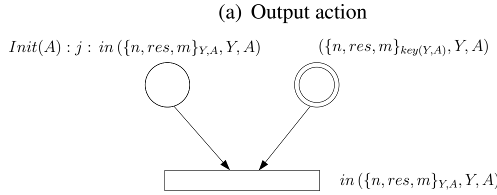
C The Events of MUTE

MUTE has three kind of events, proper to each role of the agents in the network:

Initiator Events:

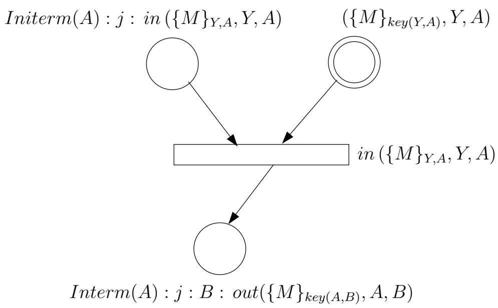


(a) Output action

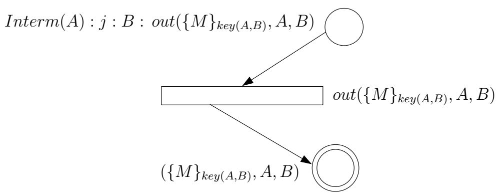


(b) Input action

Intermediator Events:

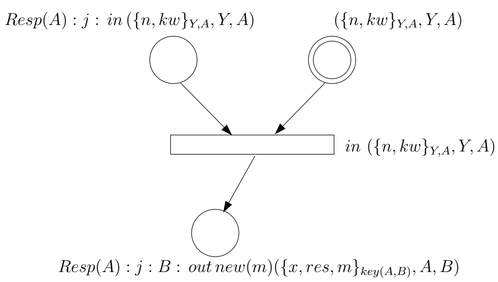


(c) Input action

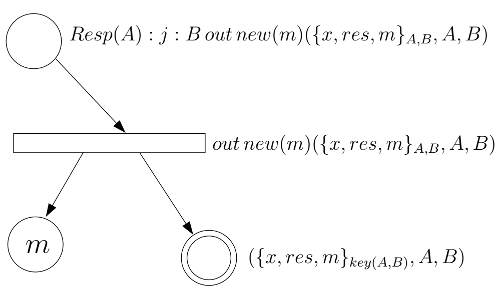


(d) Output action

Responder Events:



(e) Input action



(f) Output action

D Secrecy Property (Full Proofs)

D.1 Secrecy property for shared keys

Theorem 3.2 *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.

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

The event e_w is an event in the set

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

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, 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,these must be an earlier stage u in the run, $u < w$ st $key(A_0, B_0) \sqsubseteq t_u$ which is a contradiction. □

D.2 Secrecy property for the request

Theorem 3.3 *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

$$a_1 \longrightarrow e_v$$

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). The event e_v is an event in

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

and from the token game of nets with persistent conditions we have

$$\sigma(kw_0, e_v^o - e_{v-1}^o) \not\subseteq \{(\{n_0, kw_0\}_{key(A_0,B_0)}, A_0, B_0)\} \quad (6.1)$$

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.

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 (6.1) 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 (6.1) 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 (6.1) 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 one reaches a contradiction to property (6.1) 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 (6.1) $kw_0 = n$. By control precedence there exists an event e_u in the run st

$$e_u \longrightarrow e_v$$

and

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

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 (6.1) 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 (6.1) 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

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

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 (6.1) 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 (6.1) 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 3.2). Since this there is no possible way for a spy to reach kw_0, e_v is not a spy event. □

D.3 Secrecy property for the answer

Theorem 3.4 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

$$b_2 \longrightarrow e_v$$

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). The event e_v is an event in

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

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

$$\sigma(res_0, e_v^o - e_{v-1}^o) \not\subseteq \{(\{n_0, res_0, m_0\}_{key(A_0,B_0)}, A_0, B_0)\} \quad (6.2)$$

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.

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 (6.2) 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 (6.2) 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

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

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 (6.2) 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 (6.2) 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

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

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 (6.2) 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 (6.2) 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 3.2). Since this there is no possible way for a spy to reach kw_0, e_v is not a spy event.