Analysing the MUTE Anonymous File-Sharing System Using the Pi-calculus

Tom Chothia

CWI, Kruislaan 413, 1098 SJ, Amsterdam, The Netherlands.

Abstract. This paper gives details of a formal analysis of the MUTE system for anonymous file-sharing. We build pi-calculus models of a node that is innocent of sharing files, a node that is guilty of file-sharing and of the network environment. We then test to see if an attacker can distinguish between a connection to a guilty node and a connection to an innocent node. A *weak bi-simulation* between every guilty network and an innocent network would be required to show possible innocence. We find that such a bi-simulation cannot exist. The point at which the bi-simulation fails leads directly to a previously undiscovered attack on MUTE. We describe a fix for the MUTE system that involves using authentication keys as the nodes' pseudo identities and give details of its addition to the MUTE system.

1 Introduction

MUTE is one of the most popular anonymous peer-to-peer file-sharing systems[1]. Peers, or nodes, using MUTE will connect to a small number of other, known nodes; only the direct neighbours of a node know its IP address. Communication with remote nodes is provided by sending messages hop-to-hop across this overlay network. Routing messages in this way allows MUTE to trade efficient routing for anonymity. There is no way to find the IP address of a remote node, and direct neighbours can achieve a level of anonymity by claiming that they are just forwarding requests and files for other nodes. Every node picks a random pseudo ID that it uses to identify itself. There is a danger that an attacker may be able to link the pseudo ID and the IP address of it direct neighbours, and thus find out which files the neighbours are requesting and offering.

We analyse MUTE by building pi-calculus processes that model a node that is guilty of sharing files, a node that is innocent of sharing files (but does forward messages) and a third pi-calculus process that models the rest of the network. These processes are connected by channels, which can be bound by the pi-calculus new operator or left free to give the attacker access. We use the pi-calculus because it is expressive enough to define an accurate model of MUTE, while still being simple enough to analyse by hand or with automatic tools. There is also a large body of theoretical and implementational work to support analysis

[1] According to statistics at `http://sourceforge.net/projects/mute-net` MUTE has been downloaded over 850,000 times

in the pi-calculus. We do not make an explicit model of the attacker, rather we aim to show that for every network in which the attacker connects to a guilty node, there is another network in which the attacker connects to an innocent node, and that the two networks are indistinguishable. For this we use *weak bi-simulation*, which holds between pi-calculus processes if and only if all observable inputs and outputs of a system are the same. We show that the environment can provide "cover" for a guilty node by showing that for every guilty node G and environment E there is an innocent node I and another environment E' such that $G \mid E$ is weakly bi-similar to $I \mid E'$. In general, weak bi-simulation is necessary, but not sufficient, to show anonymity because it says nothing about how likely the actions of the two processes are. However, once a weak bi-simulation is shown to hold, we can prove that the guilty nodes have "possible innocence" [22] by showing that, assuming a fair scheduler, the matching actions in the weak bi-simulation occur with a non-negligible probability.

The contributions of this paper are the formal model of the MUTE system, the description of an attack on MUTE, found by analysing this model, and a fix for the attack. We note that this attack was only found while testing the model, the model was not built with this attack in mind, and while the checking of the model was performed by hand, it is a mechanical procedure that does not require any special insight. Furthermore, MUTE has an active development community that, in over two years of development, had not found this serious fault, as had a number of academic papers that tried to examine or extent MUTE [2, 5, 17].

There are a number of other anonymous peer-to-peer file-sharing systems [1, 28] and theoretical designs [5, 25], for more information we direct readers to our previous survey paper [7]. Also related are anonymous publishing systems such as Freenet [8] or Free Haven [11], which hide the original author of a file rather than the up loader, and the Tor middleware [10]. Bhargava and Palamidessi [4] model the dinning cryptographers protocol in the probabilistic pi-calculus [15]. They propose a new notion of anonymity that makes a careful distinction between non-deterministic and probabilistic actions. In other work Deng, Palamidessi and Pang [9] define "weak probabilistic anonymity" and use PRISM [19] to show that it holds for the dinning cryptographers protocol. Chatzikokolakis and Palamidessi further explore the definition of "probable innocence" [6]. Garcia et al. [13] develop a formal framework for proving anonymity based on epistemic logic, Schnider and Sidiropoulos [24] use CSP to check anonymity and Kremer and Ryan analyse a voting protocol in the applied pi-calculus [18]. Their approaches do not take into account the probability of observing actions.

In the next section we describe the MUTE system, then in Section 3 we review the pi-calculus. We carry out our analysis of MUTE in Section 4, with one sub-section on the pi-calculus model and another showing why we cannot get a bi-simulation. We discuss how this break down in bi-simulation can be turned into a real attack on a MUTE network in Section 5, and how MUTE can be fixed in Section 6. Finally, Section 7 concludes and suggests further work. Readers who are only interested in the attack and the fix may skip ahead to sections 5 and 6.

2 The Ants Protocol and the MUTE System

The MUTE system [23] is based on the Ant Colony Optimisation algorithm (Ants Protocol) [12, 14]. This protocol is in turn based on the way ants use pheromones when looking for food [3] and was not originally designed to keep users anonymous, rather it was designed for networks in which nodes do not have fixed positions or well-known identities. In this setting, each node has a pseudo identity that can be used to send messages to a node but does not give any information about its true identity (i.e., the node's IP address).

In order to search the network, a node broadcasts a search message with its own pseudo ID, a unique message identifier and a time-to-live counter. The search message is sent to all of the node's neighbours, which in turn send the message to all of their neighbours until the time-to-live counter runs out. Upon receiving a message a node first checks the message identity and discards any repeated messages, it then records the connection on which the message was received and the pseudo ID of the sender, in this way each node dynamically builds and maintains a routing table for all the pseudo identities it sees. To send a message to a particular pseudo ID a node sends a message with the pseudo ID as a "to ID". If a node has that pseudo ID in its routing table, it forwards the message along the most common connection. Otherwise, it forwards the message to all its neighbours. Some random rerouting can be added to allow nodes to discover new best routes, in case the network architecture has changed.

MUTE implements the Ants protocol with a non-deterministic time-to-live counter, as well as a host of other features designed to make the system efficient and user friendly. The kinds of attacker that MUTE defends against are nodes in the system that wish to link the IP address of their direct neighbours with a pseudo ID. Attackers may make as many connections to a node as they like but we do not let an attacker monitor the whole network or even all of the connections going into or out of a node; without the possibility of an unmonitored connection to an honest node the target loses all anonymity. A complete summary of a piece of software of the size of MUTE is beyond the scope of this paper, we highlight a few key features and refer the interested reader to the MUTE developer web-page for more information.

MUTE uses a complex three phase "Utility Counter" to control how far a search message travels. The first phase is equivalent to the searcher picking a time-to-live value from a long-tail distribution, which is then counted down to zero before moving to the second phase. The aim of this phase is to stop an attacker from being able to tell if their neighbour originated a search message. Once this first phase is finished the counter moves to the second phase, which is a standard time-to-live counter that counts up to 35 in increments of 7. This means that the message is forwarded for a further 5 hops. The values of 35 and 7 are used to be compatible with an older version of MUTE that used a more fine-grained, but less anonymous, version of the counter.

The third phase of the utility counter is a non-deterministic forwarding. Each node decides when it starts up how many nodes it is going to forward phase-3 messages to. A node will drop a phase-3 message with probability 3/4, and

Process $P, Q ::=$	0	The stopped process
	$\mid$ rec $a(x); P$	Input of x on channel a
	$\mid$ send $a(b)$	Output of b on channel a
	$\mid$ new $a; P$	New name declaration
	$\mid$ $P \mid Q$	P running in parallel with Q
	$\mid$ repeat$\{ P \}$	An infinite number of Ps
	$\mid$ if $(condition)$ then $\{ P \}$	Run P, if $a = b$
	$\mid$ $\prod_{j \in \{a_1, \ldots, a_n\}} P(x)$	$P(x)$ in parallel for all j

Fig. 1. The Pi-calculus Syntax

forward the message to n neighbours with probability $1/(3 \times 2^n)$. The aim of this last phase is to quickly end the search and to stop an attacker from being able to send a search message that it knows will not be forwarded to any other nodes. There must always be a chance of forwarding more copies of a search message to stop a number of attackers that are connected to the same node knowing when they have received all of the forwarded copies of a search.

All of the probabilistic choices made by a MUTE node (such as the value of the phase-1 counter or how many nodes phase-3 messages are forwarded to) are fixed when the node first starts up. This protects against statistical attacks by ensuring that the repetition of the same action yields no new information to the attacker.

MUTE's routing table stores information for the last one hundred different IDs it has seen. For each ID the routing table stores the last fifty connections over which a message from this ID was received. When either list is full the oldest entries are dropped. When a node needs to forward a message to a particular ID it randomly chooses one of the stored connections for that ID and forwards the message along that connection. If the node does not have an entry for the destination ID it sends the message to all of its neighbours.

3 The Pi-calculus

We use the asynchronous pi-calculus [16, 20] to build a formal model of the MUTE system. The pi-calculus can be thought of as a small concurrent language that is simple enough to be formally analysed while still expressive enough to capture the key elements of a system. The syntax of our version of the pi-calculus is shown in Figure 1. It is an asynchronous, late version of the calculus that includes conditional branching. We also include a product term that effectively defines families of processes. To express some of the details of the MUTE protocol, we extend the set of names to include tuples and natural numbers. Process terms are identified by a *structural congruence* "$\equiv$". The semantics of the calculus is shown in Figure 2. The labels on the arrows indicate the kind of reduction that is taking place, either an input a, an output $\bar{a}$ or an internal action τ. Output actions may carry new name declarations along with them,

$$\mathsf{send}\ a(b) \xrightarrow{\bar{a}\langle b\rangle} 0 \qquad \mathsf{rec}\ a(x); P \xrightarrow{a(x)} P$$

$$\frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q} \qquad \frac{P \xrightarrow{\nu\vec{c}.\bar{a}\langle b\rangle} P'}{\mathsf{new}\ c'; P \xrightarrow{\nu c', \vec{c}.\bar{a}\langle b\rangle} P'}$$

$$\frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\nu\vec{c}.\bar{a}\langle b\rangle} Q'}{P \mid Q \xrightarrow{\tau} \mathsf{new}\ \vec{c}; (P[b/x] \mid Q)} \quad \vec{c} \cap fn(P) = \emptyset$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad bn(\alpha) \cap fn(P) = \emptyset$$

$$\frac{P \xrightarrow{\alpha} P'}{\mathsf{new}\ a.P \xrightarrow{\alpha} \mathsf{new}\ a.P'} \quad a \notin \alpha$$

Fig. 2. Pi-calculus Semantics

indicated by the ν label. The side conditions on some of the rules ensure than the new name declaration does not accidentally capture other names. We further define $\stackrel{\alpha}{\Rightarrow}$ to be any number of internal (τ) actions followed by an α action and then another sequence of internal actions.

The first piece of syntax stop represents a stopped process, all processes must end with this term although we usually do not write it. The send $a(b)$ operation broadcasts the name b over channel a. The receive operation receives a name and substitutes it into the continuing process. The new operation creates a new communication channel. The repeat operator can perform recursion by spinning off an arbitrary number of copies of a process, $!P \equiv P \mid !P$. The bar $\mid$ represents two processes running in parallel and the match operation, if (condition) then $\{P\}$, executes P if and only if the condition holds. The product term defines the parallel composition of any number of processes $\prod_{j\in\{a_1,\ldots,a_n\}} P(x) \equiv P[a_1/x] \mid P[a_2/x] \mid \ldots \mid P[a_n/x]$. We will drop the set of names from this operator when their meaning is clear.

The semantic rules of the calculus allow two processes to communicate:

$$\mathsf{send}\ a(b) \mid \mathsf{rec}\ a(x); P \xrightarrow{\tau} P[b/x]$$

Here, the name b has been sent over the channel a and substituted for x in P. The τ on top of the arrow indicates that a communication has taken place. bn is defined as the names that are bound by a new operator and fn are the free names, i.e., those that are not bound. The side conditions on the reduction rules stop names from becoming accidentally bound by a new operator. One of the most important aspects of the pi-calculus is that new names are both new and unguessable, for instance the process new a; rec $a(x); P$ can never receive a communication on the channel a, no matter what the surrounding processes might try. For more information on the pi-calculus we refer the reader to one of the many survey papers [21].

4 Analysing MUTE in the Pi-calculus

Our formal model of MUTE comes in three pieces. We make a process that models an innocent node "I", which forwards messages and searches for files and another process that models a guilty node "G", which will also return a response to a request. A third kind of process "E", models the rest of the environment. These processes are parameterised on the communication channels that run between them. We can bind these channels with a new operator to hide them from the attacker or leave them free to allow the attacker access. The parameters also specify the probabilistic choices a node makes when it starts up, such as the value of the phase-1 counter. The behaviour of a parameterised node is non-deterministic, as oppose to probabilistic, i.e., the choice of which actions happen is due to chance occurrences that cannot be meaningfully assign a probability inside our framework, such as how often a user will search for a file.

Weak bi-simulation is often used as an equality relation between processes. Two processes are weakly bi-similar if every visible action of one process can be matched by the other process and the resulting processes are also weakly bi-similar:

Definition 1 (Weakly bi-similar). *Processes P and Q are weakly bi-similar if there exists an equivalence relation $\approx$ such that $P \approx Q$ and for all P_1 and Q_1 such that $P_1 \approx Q_1$, if $P_1 \stackrel{\alpha}{\Rightarrow} P_1'$ then:*

- *if α is an output or an internal action there exists a process Q_1' such that $Q_1 \stackrel{\alpha}{\Rightarrow} Q_1'$ and $P_1' \approx Q_1'$.*
- *if α is an input action, i.e., $\alpha = a(x)$, then for all names b, there exists a process Q_1' such that $Q_1 \stackrel{\alpha}{\Rightarrow} Q_1'$ and $P_1'[b/x] \approx Q_1'[b/x]$.*

A pi-calculus process cannot distinguish between two other pi-calculus processes that are weakly bi-similar. So for any possible pi-calculus process *Attacker* that models an attempt to break the anonymity of a node: if the two processes *A_is_Guilty* and *A_is_Innocent* are bi-similar then we know that the processes *A_is_Guilty* | *Attacker* and *A_is_Innocent* | *Attacker* are also bi-similar, so no pi-calculus attacker can discern if A is guilty.

We would like to show that a network in which the attacker can connect to a guilty node on channels $c_1, \ldots, c_i$ and to the environment on channels $c_1', \ldots, c_j'$:

$$\mathsf{new}\ c_{i+1}, .., c_k; (\ G(c_1, .., c_i, c_{i+1}, .., c_k) \mid E(c_{i+1}, .., c_k, c_1'', \ldots, c_j')\ )$$

is weakly bi-similar to a network in which an attacker can connect to an innocent node and a slightly different environment:

$$\mathsf{new}\ c_{i+1}, .., c_k; (\ I(c_1, .., c_i, c_{i+1}, .., c_k) \mid E'(c_{i+1}, .., c_k, c_1'', \ldots, c_j')\ )$$

where $c_1, \ldots, c_i$ are the private communication channels between the nodes and the environment, and $c_{i+1}, .., c_k$ are channels that the attacker can use to communicate with I and G. In the next sub-section we give the process terms for I, G and E and in the following sub-section we show that there can be no bi-simulation between guilty and innocent networks.

4.1 A Model of MUTE in the Pi-calculus

We make a number of abstractions while building our model; the aim of these simplifications is to make the model small enough to check without losing its overall correctness. These abstractions include the following:

(1) No actual files are transferred and no search keywords are used. Instead we use a "search" message that is responded to with a "reply" message.

(2) We parameterise our nodes on a fixed time-to-live value for the phase-1 counter. This value is reduced by one each hop. The counter goes to phase-2 when this value reaches zero.

(3) We simplify the nodes routing table: when forwarding a message to a particular ID the node picks any connection over which it has previously received a message from that ID. The pi-calculus does not include a concept of random access memory, so to represent storing an ID and a connection we send a pair $(id, connection)$ on a particular name. When we want to read from memory we receive on the same channel and test the id, if it matches the ID of the node we are looking for we use the connection otherwise we look for another packet. This can be thought of as using a buffered communication channel to store values.

(4) We assume that a node always knows the "to ID" of any reply message it sees. A more realistic model would have to test for this and send the reply message to all of the node's neighbours if the "to ID" is unknown.

(5) We do not use message IDs and do not drop repeated messages. We also allow a node to send a message back over the connection on which it was received, returning the message to its sender. This does not give an attacker any extra power because there is no way to limit an attacker to just one connection.

(6) In closely packed networks a node may send a request over one connection and receive the same request back over another. To simplify our model we assume that these kinds of communications do not happen.

Point 6 removes details that may reveal some information to the attacker, exactly what will depend on the arrangement of the network, we leave a fuller investigation as further work.

The results of these simplifications are that the channels pass messages with four fields:

Message format = (kind of message, the "to ID", the "from ID",
phase of the counter, value of the counter)

A message kind may be a "*search*" or a "*reply*" to a search. The from and to IDs are the pseudo IDs of the originator of the message and its final destination (not the IDs of the nodes that the message is being past between on the current hop). Search messages are broadcast to the network and so do not use the "to ID" field, in this case we will write "none" as the "to ID".

The processes are parameterised on the communication channels they will use to talk to the environment and the attacker. To stop a node communicating with itself we represent communication streams as a pair $c_j = \langle c_j^i, c_j^o \rangle$. The node will use the c_j^i channel for input and the c_j^o channel for output. In order not to clutter our process terms, we will write "$\mathsf{new}\ c_j$" for "$\mathsf{new}\ c_j^i, c_j^o$".

$I(connections, forward, ttl)$
$\equiv$ new $my\_id, memory; I_{ID}(my\_id, connections, forward, ttl, memory)$

$I_{ID}(my\_id, \langle\ \langle c_1^i, c_1^o\rangle, .., \langle c_n^i, c_n^o\rangle\ \rangle, \langle\ \langle c_{for\_1}^i, c_{for\_1}^o\rangle, .., \langle c_{for\_p}^i, c_{for\_p}^o\rangle\ \rangle, ttl, memory)$
$\equiv \Pi_j$ repeat { rec $c_j^i(kind, to\_id, from\_id, phase, counter)$;
send $memory(from\_id, c_i^o)$
| if $(kind = search)$ then { $FORWARDMESSAGE$ }
| if $(kind = reply)$ then { $REPLY$ }
}
| repeat { Π_l send $c_l^o(search, none, my\_id, 1, ttl)$}

$FORWARDMESSAGE \equiv$
if $(phase = 1$ and $counter > 1)$ then
$\{\Pi_k$ send $c_k^o(kind, to\_id, from\_id, 1, counter - 1)\}$
| if $(phase = 1$ and $counter = 1)$ then
$\{\Pi_k$ send $c_k^o(kind, to\_id, from\_id, 2, 0)\}$
| if $(phase = 2$ and $counter < 35)$ then
$\{\Pi_k$ send $c_k^o(kind, to\_id, from\_id, 2, counter + 7)\}$
| if $(phase = 2$ and $counter \geq 35)$ then
$\{\Pi_k$ send $c_{for\_k}^o(search, to\_id, from\_id, 3, 0)\}$
| if $(phase = 3)$ then $\{\Pi_k$ send $c_{for\_k}^o(search, to\_id, from\_id, 3, 0)\}$ }

$REPLY \equiv$ if $(to\_id = my\_id)$ then {stop}
| if $(to\_id \neq my\_id)$ then
{ new $loop$; send $loop$;
repeat{ rec $loop$; rec $memory(x, channel)$; (send $memory(x, channel)$
| if $(x \neq to\_id)$ then {send $loop$}
| if $(x = to\_id)$ then
{send $channel(kind, to\_id, from\_id, phase, counter)$}) }

Fig. 3. An Innocent Node

The process term for the innocent node is given in Figure 3. The node's parameters are defined as follows:

I(a tuple of connections to other nodes,
a tuple of the connections on which the node will forward phase 3 messages,
the initial time-to-live value used for phase-1 when generating a search message)

We define I in terms of another process I_{ID} that also states the node's ID and the channel name it uses for memory. The I_{ID} process listens repeatedly, for a message on any of its input channels. When it receives a message it stores the message's "from ID" and the channel on which the message was received by sending them on the memory channel. The node then tests the message's kind to see if it is a search message or a reply message. If the message is a search message the utility counter is tested and altered, and the message is forwarded.

$G(connections, forward, ttl)$
$\equiv$ new $my\_id, memory; G_{ID}(my\_id, connections, forward, ttl, memory)$

$G_{ID}(my\_id, \langle\ \langle c_1^i, c_1^o\rangle, .., \langle c_n^i, c_n^o\rangle\ \rangle, \langle c_{for}^i, c_{for}^o\rangle, ttl, memory) \equiv$
Π_j repeat{rec $c_j^i(kind, to\_id, from\_id, phase, counter)$;
send $memory(from\_id, c_i^o)$
| if$(kind = search)$ then
{new $loop$; send $loop$;
repeat{rec $loop$; rec $memory(x, channel)$; (send $memory(x, channel)$
| if $(x \neq from\_id)$ then {send $loop$}
| if $(x = from\_id)$ then
{send $channel(reply, from\_id, my\_id, none, none)$}) }
| $FORWARDMESSAGE$ }
| if $(kind = reply)$ then { $REPLY$ } }
| repeat { Π_l send $c_l^o(search, none, my\_id, 1, ttl)$}

Fig. 4. A Guilty Node

If the message received by the node is a reply message then the node checks to see if the message is addressed to itself. If it is then this thread of the process stops and the node continues to listen on its connections (this test and stop has no functionality and is included for clarity). If the reply is addressed to another node then the process looks up the "to ID" in its memory and forwards the message. The last line of the I_{ID} process allows the node to send search messages to any of its neighbours.

The process term for a guilty node is given in Figure 4. This process works in the same way as the innocent node except that, after receiving any search message, the node looks up the "from ID" and returns a reply.

Figure 5 contains a pi-calculus model of the environment which has one connection to a node. The process $E(c, n, j)$ models an environment that communicates over the channel c and includes n individual nodes that may search for files, out of which j nodes will reply to requests for files. The parameters of the E_{ID} process include the IDs of its nodes and meta-functions that map each ID to a utility counter value. There is not a true structural equivalence between E and E_{ID} because the meta-functions are not really part of the calculus, but rather a device that allows us to define a family of processes.

In Figure 6 we expand this to the environment with m connections by overloading E and E_{ID}. This process uses a tuple of values for each of the single values in the one connection environment. For i, ranging from 1 to m, the process uses the channel c_i over which n_i individual nodes may search for files. These nodes use the IDs $id\_i_1, \ldots, id\_i_{n_i}$ and the first j_i of these will reply to requests for files.

$$E(c,n,j) \cong \mathsf{new}\ id_1,\ldots,id_n; E_{ID}(c,\langle id_1,\ldots,id_n\rangle,j,f_p,f_c)$$

$$\begin{aligned}
&E_{ID}(\langle c^i,c^o\rangle,\langle id_1,\ldots,id_n\rangle,j,f_p,f_c)\\
&\quad\equiv \mathsf{repeat}\{\mathsf{rec}\ c^i(kind,to_id,from_id,phase,counter);\\
&\qquad\quad \mathsf{if}(kind=search)\ \mathsf{then}\\
&\qquad\qquad \{\Pi_{i\in\{1,\ldots,j\}}\ \mathsf{send}\ channel(reply,from_id,id_j,0,0);\}\ \}\\
&\qquad \mid \Pi_i\ \mathsf{repeat}\ \{\ \mathsf{send}\ c^o(search,none,id_i,f_p(id_j),f_c(id_j))\ \}
\end{aligned}$$

Fig. 5. The Environment with one connection

$$\begin{aligned}
&E_{ID}(\langle\langle c_1^i,c_1^o\rangle,\ldots,\langle c_m^i,c_m^o\rangle\rangle,\langle n_1,\ldots,n_m\rangle,\langle j_1,\ldots,j_m\rangle)\\
&\quad\cong \mathsf{new}\ (id_1_1,\ldots,id_1_{n_1}),(id_2_1,\ldots,id_2_{n_2}),\ldots,(id_m_1,\ldots,id_m_{n_m});\\
&\qquad E_{ID}(\langle\langle c_1^i,c_1^o\rangle,\ldots,\langle c_m^i,c_m^o\rangle\rangle,\langle j_1,\ldots,j_m\rangle,\\
&\qquad\quad \langle\langle id_1_1,\ldots,id_1_{n_1}\rangle,\langle id_2_1,\ldots,id_2_{n_2}\rangle,\ldots,\langle id_m_1,\ldots,id_m_{n_m}\rangle\rangle,\\
&\qquad\quad \langle f_1^p,\ldots,f_m^p\rangle,\langle f_1^c,\ldots,f_m^c\rangle\ )\\
&E_{ID}(\ldots)\equiv \Pi_{k\in\{1,..,m\}}\mathsf{repeat}\{\mathsf{rec}\ c_k^i(kind,to_id,from_id,phase,counter);\\
&\qquad\qquad\qquad\qquad \mathsf{if} i(knd=search)\ \mathsf{then}\\
&\qquad\qquad\qquad\qquad\quad \{\Pi_{i\in\{1,\ldots,j_k\}}\ \mathsf{send}\ channel(reply,from_id,id_k_j,0,0);\}\ \}\\
&\qquad\qquad \mid \Pi_{k\in\{1,..,m\}}\Pi_{i\in\{1,..,n_k\}}\mathsf{repeat}\{\mathsf{send}\ c_k^o(search,none,id_k_i,f_m^p(id_k_i),f_m^c(id_k_i))\}
\end{aligned}$$

Fig. 6. The Environment with m connections

4.2 No Bi-Simulations Between Guilty and Innocent Nodes

The environment should provide "cover" for guilty nodes. As an example, consider the case in which a guilty node has one connection to the attacker and one connection to the environment:

$$\mathsf{new}\ c_2; G(\langle c_1,c_2\rangle,\langle c_2\rangle,m)\mid E(c_2,n,j)$$

The only communication channel in this process that is not new is the c_1 channel, therefore this is the only channel that the attacker may use to communicate with the process. The guilty node will reply to a search request sent on c_1, and so expose its own ID. The environment will do likewise and expose its IDs along with their utility counter values:

$$\begin{aligned}
&\mathsf{new}\ c_2; G(\langle c_1,c_2\rangle,\langle c_2\rangle,m)\mid E(c_2,n,j)\\
\cong\ &\mathsf{new}\ c_2; G(\langle c_1,c_2\rangle,\langle c_2\rangle,m)\mid \mathsf{new}\ id_1,\ldots,id_n; E(c_2,\langle id_1,\ldots,id_n\rangle,j,f_p,f_c)\\
\stackrel{\vec{\alpha}}{\Rightarrow}\ &\mathsf{new}\ c_2,mem; G_{ID}(g_id,\langle c_1,c_2\rangle,\langle c_2\rangle,m,mem)\mid \mathsf{send}\ mem(id_1,c_2)\ldots\\
&\qquad\mid \mathsf{send}\ mem(id_j,c_2)\mid \mathsf{new}\ id_{j+1},\ldots,id_n; E(c_2,\langle id_1,\ldots,id_n\rangle,j,f_p,f_c)\ )
\end{aligned}$$

where the $\vec{\alpha}$ actions are the search input from the attacker followed by some permutation of the responses on channel c_1.

The anonymity should come from the fact that the same observations may come from an innocent node and an environment that provides one more response with a phase-1 utility counter set to one higher than m. We can verify that this innocent network can perform the same $\vec{\alpha}$ actions as the guilty network:

$$
\begin{array}{ll}
 & \mathsf{new}\ c_2; I(\langle c_1, c_2\rangle, \langle c_2\rangle, m) \mid E(c_2, n, j+1) \\
\cong & \mathsf{new}\ c_2; I(\langle c_1, c_2\rangle, \langle c_2\rangle, m) \mid \mathsf{new}\ id_1, \ldots, id_n; E(c_2, \langle id_1, \ldots, id_n\rangle, (j+1), f'_p, f'_c) \\
\stackrel{\vec{\alpha}}{\Rightarrow} & \mathsf{new}\ c_2, mem, i\_id; I_{ID}(i\_id, \langle c_1, c_2\rangle, \langle c_2\rangle, m, mem) \mid \mathsf{send}\ mem(id_1, c_2) \ldots \\
 & \quad \mid \mathsf{send}\ mem(id_{j+1}, c_2) \mid \mathsf{new}\ id_{j+2}, \ldots, id_n; E(c_2, \langle id_1, \ldots, id_n\rangle, j+1, f'_p, f'_c)\ )
\end{array}
$$

where $f'_p(id_{j+1} = 1), f'_c(id_{n+1}) = m + 1$. So, in order to be able to show some level of anonymity, we would like to show the following bi-simulation:

$$
\begin{array}{r}
\mathsf{new}\ c_2; G(\langle c_1, c_2\rangle, \langle c_2\rangle, m) \mid E(c_2, n, j) \\
\approx \mathsf{new}\ c_2; I(\langle c_1, c_2\rangle, \langle c_2\rangle, m) \mid E(c_2, n, j+1)
\end{array}
$$

where $j < n$, i.e., there are both innocent and guilty nodes in the environment.

Upon examination we find that relations of this kind do not hold. Let us start with P_G and P_I as follows:

$$
\begin{array}{c}
P_G \equiv \mathsf{new}\ c_2; (\ G(\langle c_1, c_2\rangle, \langle c_2\rangle, m) \mid E(c_2, n, j)\ ) \\
P_I \equiv \mathsf{new}\ c_2; (\ I(\langle c_1, c_2\rangle, \langle c_2\rangle, m) \mid E(c_2, n, j+1)\ )
\end{array}
$$

We follow a fixed method to check the bi-similarity of two processes. We enumerate all of the inputs, outputs and internal actions that a process can perform and then look for matching actions in the other process. We then test each pair of resulting processes for bi-similarity. Most possible actions will not produce interesting reactions, for instance any node will discard messages that do not use "search" or "reply" as its kind. Ultimately the processes may loop around to processes that we already consider bi-similar, in which case we proceed to check the other processes. If we find processes that do not match we backtrack and see if there were any other possible matchings of actions that would have worked. If processes are to complicated to effectively check by hand they can be checked with automated tools such as the Mobility Workbench [27] or Level 0/1 [26].

One of the first actions that we must check for P_G and P_I is the input of a well-formed search message. In response both processes return equivalent reply messages. In the P_G network these include the reply from the guilty note. The innocent network can match these outputs with replies from the environment. At this point the IDs are public and the node has recorded the IDs from the environment:

$$
\begin{array}{c}
P_G \stackrel{\vec{\alpha}_1}{\Rightarrow} \mathsf{new}\ c_2, mem; (\ G_{ID}(n\_id, \langle c_1, c_2\rangle, \langle c_2\rangle, m, mem) \mid \mathsf{send}\ mem(id_1, c_2) \ldots \\
\quad \mid \mathsf{send}\ mem, (id_j, c_2) \mid E(c_2, \langle id_1, \ldots, id_n\rangle, j, f'_p, f'_c)\ ) \\
P_I \stackrel{\vec{\alpha}_1}{\Rightarrow} \mathsf{new}\ c_2, mem; (\ I_{ID}(n\_id, \langle c_1, c_2\rangle, \langle c_2\rangle, m, mem) \mid \mathsf{send}\ mem(id_1, c_2) \ldots \\
\quad \mid \mathsf{send}\ mem(id_{j+1}, c_2) \mid E(c_2, \langle id_1, \ldots, id_n\rangle, j+1, f'_p, f'_c)\ )
\end{array}
$$

The free names that are available to the attacker in P_G now include $id_1, \ldots, id_j$ and n_id and in P_I they include $id_1, \ldots, id_j, id_{j+1}$. As these are all new names the attacker cannot distinguish between these two sets ... yet.

We must now check to see if the two processes behave in the same way when they receive inputs that use the names that have just been outputted. For the guilty network one message that must be checked is $\mathsf{send}\ c_1(search, none, n\_id, 1, 7)$, i.e., a search message with the guilty node's ID as the "from ID". As n_id is just one of a set of free names this action can be matched by the innocent network with a similar message using an ID from the environment, id_k say, resulting in the processes:

$$\begin{aligned} P_G \stackrel{\vec{\alpha}_3}{\Rightarrow} & \ \mathsf{new}\ c_2, mem; G_{ID}(n\_id, \langle c_1, c_2\rangle, \langle c_2\rangle, m, mem) \\ & \quad \mid \mathsf{send}\ mem(n\_id, c_1) \mid \mathsf{send}\ mem(id_1, c_2) \mid \ \ldots \\ & \quad\quad \mid \mathsf{send}\ mem(id_j, c_2) \mid E(c_2, \langle id_1, .., id_n\rangle, j, f'_p, f'_c)\ ) \\ P_I \stackrel{\vec{\alpha}_3}{\Rightarrow} & \ \mathsf{new}\ c_2, mem; I_{ID}(n\_id, \langle c_1, c_2\rangle, \langle c_2\rangle, m, mem) \\ & \quad \mid \mathsf{send}\ mem(id_k, c_1) \mid \mathsf{send}\ mem(id_1, c_2) \mid \ \ldots \\ & \quad\quad \mid \mathsf{send}\ mem(id_{j+1}, c_2) \mid E(c_2, \langle id_1, .., id_n\rangle, j+1, f'_p, f'_c)\ ) \end{aligned}$$

For P_G and P_I to be bi-similar these processes must be bi-similar. Both of them can indeed perform the same inputs and outputs. Including the input of a reply message address to the ID that has just been used for the search message, i.e., $\mathsf{send}\ c_1(reply, n\_id, a_{id}, 0, 0)$ for P_G and $\mathsf{send}\ c_1(reply, id_k,\ a_{id}, 0, 0)$ for P_I The innocent node looks up the ID, it may find the c_1 connection and return the request on c_1 as output. The guilty node, on the other hand, recognises its own ID and accepts the message. It cannot perform the same output as the innocent node and therefore P_G and P_I cannot be bi-similar.

If we examine the actions that lead to these un-similar processes we can see that the attacker tried to "steal" one of the IDs that it has seen as a "from ID" of a reply to its message. The attacker can then use this process to test IDs to see which one belongs to its neighbour because, out of all the IDs in all the reply messages that an attacker may see, the guilty node's ID is the only one that cannot be stolen.

5 Description of the Attack on MUTE

The difference between the processes in the pi-calculus model was that the innocent node might be able to perform an action, whereas the guilty node could not. To build a real attack out of this we must force the innocent node to perform the action so the guilty node will stand out. The idea behind the real attack is that the attacker can "steal" an ID by sending fake messages using the target ID as the "from ID". If it sends enough messages then its neighbouring nodes will forward messages addressed to the target ID over their connection with the attacker. One exception to this is if the ID the attacker is trying to steal belongs to the neighbour, as the neighbour will never forward messages addressed to itself. Therefore the attacker can use this method of stealing IDs to test any IDs it sees, if the ID cannot be stolen then the ID belongs to the neighbour.

We saw in Section 2 that MUTE looks at the last fifty messages when deciding where to route a message. Only IDs that are seen on search messages with phase-1 counters are possibilities for the neighbours ID and only search messages with

phase-3 counters can be dropped. Therefore, if the attacker sees some messages with a phase-1 counter and others reach the neighbour with a phase-3 counter and are dropped, we know that the messages that are dropped must be slower and so they will not affect the routing table. This means that if the attacker can send fifty messages with the target ID to its neighbour, without any messages with that ID coming from the neighbour, then the attacker will receive any messages send to that ID via the target node, unless the ID belongs to the target node. There is still a small possibility that the neighbour is receiving or forwarding a file from the real owner of the ID, in which case the large number of messages that the neighbour is receiving might mean the attacker fails to steal an address that does not belong to the target node. To avoid this possibility the attack can be repeated at regular intervals.

The attack on MUTE would run as follows:

1. The attacker makes two connections to the target node, monitors these connections and selects the "from ID" with the highest time-to-live counter.
2. The attacker forms new search messages using the selected ID as the "from ID" and repeatedly sends them to the neighbour until it has sent fifty messages without receiving any messages from the ID it is trying to steal.
3. The attacker then sends a reply message addressed to the selected ID along its other connection with the target node. If the message is not sent back to the attacker then it is likely that the target ID belongs to the neighbour.
4. These steps can be repeated at regular intervals to discount the possibility that the neighbour is receiving or forwarding a file from the target ID.
5. If the attacker receives the message back then the selected ID does not belong to the target node, so the attacker must select another ID and start again.
6. If the neighbour still does not bounce the message back to the attacker then, with a high degree of probability, the attacker has found the neighbour's ID and the attacker can then find out what files the neighbour is offering.

6 Fixing MUTE

This attack is made possible by the Ants Protocol's adaptive routing system and the fact that nodes will never forward messages addressed to themselves. Key to the success of the attack is the attacker's ability to corrupt its neighbour's routing table in a known way. This in turn is only possible because the attacker can fake messages with another node's ID.

We can solve this problem by stopping the attacker from being able to forge messages. This can be done by having all nodes start by generating an authentication and signature key, from any suitably secure public key encryption scheme. The nodes can then use the authentication keys as their IDs. This authentication key would be used in exactly the same way as the node's ID. However, each node would also sign the message ID. When any node receives a message, it can check the signed message ID using the "from ID". As the attacker cannot correctly sign the message ID it can no longer forge messages. Such a scheme also benefits

from a fair degree of backward compatibility. Older nodes need not be aware that the ID is also an authentication key. The checking is also optional; nodes may choose to only check messages it they spot suspicious activity.

The level of popularity enjoyed by any system that offers anonymity to the user will be partly based on the level of trust potential users place in these claims. To maintain a level of trust in the MUTE system it is important to implement this fix before the flaw is widely known. With this in mind we sent an early copy of this paper to the lead programmer on the MUTE project. They were happy to accept the results of the formal analysis and agreed to implement the fix suggested above. To remain compatible with earlier versions of the client the pseudo IDs could not be longer than 160-bits, which is too short for a RSA public key. We briefly considered using an elliptic curve digital signature algorithm that would allow for public keys of this length, but the use of less well known cryptographic algorithms proved unpopular.

The final solution was to use a SHA1 hash of a 1024-bit RSA authentication key as the pseudo ID and include the authentication key in the message header, along with the signed message ID. This would require changing the message header from random a "From ID" and "Message ID" to a 1024-bit RSA authentication key, the SHA1 hash of that key as the "From ID", along with the signed message ID. It was also found that nodes would only store message IDs for a few hours so to avoid replay attacks a counter based timestamp was included in the signature of the message. This solutions was added to the 0.5 release of MUTE; the C++ source code for the implementation is available at `http://mute-net.cvs.sourceforge.net`[2] .

7 Conclusion and Further Work

We have modelled the MUTE system in the pi-calculus and we have shown that it is not possible to have a bi-simulation between every network in which the attacker connects to a guilty node and a network in which the attacker connects to an innocent node. The point at which this bi-simulation fails leads to an attack against the MUTE system. The attack, which involves "stealing" the name of another node, is a serious problem that compromises the anonymity of any node that neighbours an attacker. We suggested a fix for this attack based on using an authentication key as the node's pseudo ID.

Our general methodology was to try to show that the environment could provide cover for any guilty node. In particular that for all parameters p_g, p_e there exists some other parameters p_i, p'_e such that:

$$\text{Guilty node}(p_g) \mid \text{Environment } (p_e) \approx \text{Innocent node } (p_i) \mid \text{Environment } (p'_e)$$

We hope that this method can be used to prove the anonymity of other systems in which the environment provides cover for guilty nodes.

[2] In the first five months after its release the patched version of MUTE was downloaded over 75,000 times

As further work we hope to be able to prove that some form of the Ants protocol does provide anonymity. If we do not allow inputs to our model that use IDs that have been observed as outputs we can show for every guilty network there is a bi-similar innocent network. However a true correctness result will require a more realistic model of the environment. We would expect a node not to be anonymous when connected to some small, pathological environments. So it would be necessary to find out what kind of environments provide adequate cover for a node.

Acknowledgement

We would like to thank the Comète Team at the École Polytechnique for many useful discussions about anonymity, and especially Jun Pang for comments on an early draft of this paper. We would also like to thank Jason Rohrer, who is responsible for creating MUTE and who implemented the fix described in this paper.

References

1. Ants p2p, http://antsp2p.sourceforge.net/, 2003.
2. Andres Aristizabal, Hugo Lopez, Camilo Rueda, and Frank D. Valencia. Formally reasoning about security issues in p2p protocols: A case study. In *Third Taiwanese-French Conference on Information Technology*, 2005.
3. R. Beckers, J. L. Deneubourg, and S. Goss. Trails and u-turns in the selection of the shortest path by the ant lasius niger. *Journal of Theoretical Biology*, 159:397–415, 1992.
4. Mohit Bhargava and Catuscia Palamidessi. Probabilistic anonymity. In *CONCUR, LNCS 3653*, pages 171–185, 2005.
5. Steve Bono, Christopher A, Soghoian, and Fabian Monrose. Mantis: A high-performance, anonymity preserving, p2p network, 2004. Johns Hopkins University Information Security Institute Technical Report TR-2004-01-B-ISI-JHU.
6. Konstantinos Chatzikokolakis and Catuscia Palamidessi. Probable innocence revisited. In *Formal Aspects in Security and Trust, LNCS 3866*, pages 142–157, 2005.
7. Tom Chothia and Konstantinos Chatzikokolakis. A survey of anonymous peer-to-peer file-sharing. In *EUC Workshops, LNCS*, pages 744–755, 2005.
8. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *LNCS*, 2009:46+, 2001.
9. Y. Deng, C. Palamidessi, and J. Pang. Weak probabilistic anonymity. In *Proc. 3rd International Workshop on Security Issues in Concurrency (SecCo'05)*, 2005.
10. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.
11. Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

12. Marco Dorigo and Gianni Di Caro. The ant colony optimization meta-heuristic. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.
13. Flavio D. Garcia, Ichiro Hasuo, Wolter Pieters, and Peter van Rossum. Provable anonymity. In *Proceedings of the 3rd ACM Workshop on Formal Methods in Security Engineering (FMSE05)*, 2005.
14. Mesut Gunes, Udo Sorges, and Imed Bouazzi. Ara – the ant-colony based routing algorithm for manets. In *Proceedings of the International Workshop on Ad Hoc Networking (IWAHN 2002)*, Vancouver, August 2002.
15. Oltea Mihaela Herescu and Catuscia Palamidessi. Probabilistic asynchronous pi-calculus. In *Foundations of Software Science and Computation Structure*, pages 146–160, 2000.
16. Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *European Conference on Object-Oriented Programming*, LNCS, pages 133–147, 1991.
17. Byung Ryong Kim, Ki Chang Kim, and Yoo Sung Kim. Securing anonymity in p2p network. In *sOc-EUSAI '05: Proceedings of the joint conference on Smart objects and ambient intelligence*. ACM Press, 2005.
18. Steve Kremer and Mark D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Proceedings of the 14th European Symposium on Programming (ESOP'05)*, LNCS, pages 186–200, 2005.
19. M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. In *Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume LNCS 2324, pages 200–204, 2002.
20. Robin Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, volume 94 of *Computer and Systems Sciences*, pages 203–246. 1993.
21. Joachim Parrow. *Handbook of Process Algebra*, chapter An Introduction to the pi-calculus. Elsevier, 2001.
22. M. Reiter and A. Rubin. Crowds: anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.
23. Jason Rohrer. Mute technical details. http://mute-net.sourceforge.net/technicalDetails.shtml, 2006.
24. Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In *ESORICS*, pages 198–218, 1996.
25. Emin Gun Sirer, Sharad Goel, Mark Robson, and Doan Engin. Eluding carnivores: File sharing with strong anonymity, 2004. Cornell Univ. Tech. Rep.
26. Alwen Tiu. Level 0/1 prover: A tutorial. Avilable online at: http://www.lix.polytechnique.fr/$\tilde{}$tiu/lincproject/, 2004.
27. Björn Victor and Faron Moller. The Mobility Workbench — a tool for the π-calculus. In *CAV'94: Computer Aided Verification*, LNCS 818, pages 428–440, 1994.
28. Waste, http://waste.sourceforge.net/, 2003.