Chaining Secure Channels

Christopher Dilloway
Oxford University Computing Laboratory

May 29, 2008

Abstract

Security architectures often make use of secure transport protocols to protect network messages: the transport protocols provide secure channels between hosts. In this report we examine the possibilities for chaining secure channels. We present a surprising theorem that shows that, in some cases, two channels can be chained through a proxy to produce a stronger channel. We also show that the channel established through a proxy is at least as strong as the greatest lower bound of the channels established to and from the proxy.

1 Background

A popular technique for designing a security architecture is to rely on a secure transport layer to protect messages on the network, and provide secure channels between different hosts; see e.g. [OAS05, Vis06, WSF+03]. In such circumstances it is important to understand what is required of the secure transport protocol, and, conversely, what services are provided by different protocols.

In [DL08] we describe a framework for specifying the sorts of security guarantees that might be provided by secure channels. We capture security properties using CSP-style trace specifications, building on the work of Broadfoot and Lowe [BL03]. Our formalism allows us to compare the strengths of different secure channels: if an architecture is correct when it uses a particular secure channel, it will still be correct when it uses a stronger channel.

In [DL08] we exclusively describe channels that secure point-to-point connections; in this report we examine the possibilities for chaining secure channels. We consider chaining channels in two different ways: first through a set of dedicated intermediaries (simple proxies), and then through a (much smaller) set of trustworthy (multiplexing) proxies. We present a surprising theorem that shows how, under some circumstances, two channels can be chained to produce a stronger channel. We also show that the channel established through a proxy is always simulated by (i.e. is at least as strong

as) the greatest lower bound of the channels established to and from the proxy.

The results presented in this report are particularly relevant to real-world use of secure channels. Many organisations arrange their computers in a trusted intranet, and only allow external access through a proxy. For example: grid architectures may only allow communication to the servers that the grid is comprised of through a gatekeeper, and network firewalls may scan all traffic that passes through them, and automatically block messages that contain viruses. In order to perform these roles, the gatekeeper and firewall must act as proxies for the trusted servers.

It is not obvious that a secure connection is established between two agents by establishing two secure connections through a trusted proxy. It is also not obvious which security properties, if any, are provided by establishing secure channels through a proxy. The aim of this paper, therefore, is to investigate these sorts of chains of channels in order to determine which security properties the overall channel can provide.

In Sections 2 and 3 we describe the model of secure channel specifications from [DL08]. We define a set of valid system traces; secure channel specifications are then given as predicates over this set. In Section 4 we describe the simulation relation from [DL08] which captures a partial order over channel specifications; we use this relation to define an equivalence relation.

In Section 5 we describe the first chaining theorem; in this section the proxies are dedicated intermediaries. For every pair of agents there is a unique simple proxy whose only job is to forward messages from the first agent to the second. Every agent knows, and can reliably verify, the identity of their own proxies, and also of the proxies who send messages to them on other agents' behalf. This public knowledge of the job of each simple proxy restricts the intruder's behaviour, and so, in some cases, two channels can be chained to produce a stronger channel.

In Section 6 we describe the second chaining theorem; in this section the proxies are more general. Each proxy accepts messages from any agent, and is willing to forward them to any other agent. The agents and the proxies add extra information to the application-layer messages that they send in order to identify the third party involved in the message (the original sender, or the intended recipient). This extra information limits the intruder's behaviour, and so, as before, two channels can be chained to produce a stronger channel.

Finally, in Section 7 we discuss related work, and in Section 8 we conclude, we highlight the utility of the results presented in this report, and we discuss future work.

2 Secure channels

In [DL08] we define an abstract network in terms of honest agents, who send and receive messages, and an intruder, who has several events he can use to manipulate the messages being passed on the network, and who can also send and receive messages. The model reflects the traditional internet protocol stack, but we add a new layer between the transport layer and the application layer: the secure transport layer. We abstract all of the layers beneath the secure transport layer into a network layer.

In the application layer the agents play roles, establish channels, and send and receive messages over these channels. The agents refer to the channels by local connection identifiers; these can be thought of as *handles* to the communication channels. The secure transport layer contains *protocol agents*, which translate the higher level events into lower level events (e.g. by encrypting or signing messages), and vice versa (e.g. by decrypting messages or verifying signatures).

A channel is described by an ordered pair of roles; for example, the channel (R_i, R_j) (which we often write as $R_i \rightarrow R_j$) is the channel in which agents playing role R_i send messages to agents playing role R_j.

We treat encryption formally. All messages are drawn from the message space, *Message*, which is partitioned into two sets: application-layer messages ($Message_{App}$) and transport-layer messages ($Message_{TL}$). We assume a relation $\vdash$ defined over this type: for $X \subseteq Message$, and $m : Message$, $X \vdash m$ means that m can be deduced from the set X. We assume that the relation $\vdash$ is monotonic and transitive. Often in our examples we use the deduction rules from [RSG+01] which model Dolev-Yao style symbolic encryption [DY83]: the intruder can only read messages he has the decryption keys for, and can only create encrypted (or signed) messages when he knows the requisite keys. We assume that the intruder has some initial knowledge $IIK \subseteq Message$.

We use the following events, where m ranges over the set $Message_{App}$ of application-layer messages.

send$.(A, R_i).c_A.(B, R_j).m$: the agent (A, R_i) sends message m, intended for agent (B, R_j), in a connection identified by A as c_A.

receive$.(B, R_j).c_B.(A, R_i).m$: the agent (B, R_j) receives message m, apparently from agent (A, R_i), in a connection identified by B as c_B.

fake$.(A, R_i).(B, R_j).c_B.m$: the intruder fakes a *send* of message m to agent (B, R_j) in connection c_B with the identity of honest agent (A, R_i).

hijack$.(A, R_i) \rightarrow (A', R_i).(B, R_j) \rightarrow (B', R_j).c_{B'}.m$: the intruder modifies a previously sent message m and changes the sender from (A, R_i) to (A', R_i), and the receiver from (B, R_j) to (B', R_j) so that B' accepts it in connection $c_{B'}$.

By changing both, just one, or neither of the identities associated with a message, the intruder can use the *hijack* event in four different ways: to

replay a previously-sent message, to *re-ascribe*[1] a message, to *redirect* a message, or to *re-ascribe and redirect* a message.

For example, if application layer message m from A to B is encoded as the transport layer message $A, \{m\}_{PK(B)}$, where $PK(B)$ is B's public key, then a dishonest agent may re-ascribe this message, replacing the identity A with an arbitrary other identity. On the other hand, if m is encoded as $\{\{m\}_{PK(B)}\}_{SK(A)}$, where $SK(A)$ is A's secret key, then the intruder can only re-ascribe it by replacing the signature with his own: he can only do so with a dishonest identity. Recall that the intruder can only fake messages that he knows, so in both the above cases, the intruder could not have used a *fake* event, except if he happened to know m.

Likewise, if m is encoded as $B, \{m\}_{SK(A)}$, then a dishonest agent may redirect this message, replacing the identity B with an arbitrary other identity. On the other hand, if m is encoded as $\{\{m\}_{SK(A)}\}_{PK(B)}$, then the intruder can redirect it only if he possesses $SK(B)$: he can only redirect messages sent to him. Note that the intruder could not have used a *fake* event, because he cannot choose the value of m.

In [DL08] we specify four rules that define the application-layer behaviour accepted by our networks. These rules are not intended to capture channel properties; rather, they define some sanity conditions in order to remove artificial and irrelevant behaviour from our networks.

$\mathcal{N}_1$ The intruder never sends or fakes messages to himself, never fakes messages with a dishonest identity, and never redirects a message sent to himself and re-ascribes it with his own identity;

$\mathcal{N}_2$ The intruder can only hijack messages that were previously sent;

$\mathcal{N}_3$ The intruder can only send or fake messages that he knows; after any trace, the intruder's knowledge (with initial knowledge IIK) is described by the function $IntruderKnows_{IIK}(tr)$;

$\mathcal{N}_4$ No agent may receive a message that was not previously sent, faked or hijacked to them.

We also specify two rules that define the relationship between the application-layer events and the transport-layer events performed by the honest agents:

$\mathcal{A}_1$ the transport-layer messages sent by an honest agent's protocol agent are a prefix of (a suitable decoding[2] of) the application-layer messages sent by the agent;

[1] To ascribe means to attribute a text to a particular person; hence we use "re-ascribe" to describe the intruder's activity when he changes the identity of the sender of a message.

[2] We assume the existence of a partial, symbolic decoding function that transforms traces of transport-layer send and receive events on a single connection into traces of application-layer send and receive events on that connection.

$\mathcal{A}_2$ the application-layer messages received by an honest agent are a prefix of (a suitable decoding of) the transport-layer messages received by their protocol agent.

The intruder has additional capabilities: as well as sending and receiving application-layer messages, he can add transport layer messages to the network (put_{TL}) or remove them from it (get_{TL}). In general, we expect that a *hijack* event will usually be preceded by a get_{TL} event and followed by a put_{TL} event; similarly, we expect a *fake* event to be followed by a put_{TL} event.

We specify channels by giving trace specifications. The set of valid system traces is the (prefix-closed) set of traces that satisfy properties $\mathcal{N}_1$–$\mathcal{N}_4$ and $\mathcal{A}_1$–$\mathcal{A}_2$; the set is dependent upon the intruder's initial knowledge: the greater this set, the more traces are valid (as the intruder knows, and so can send, more messages).

A channel specification is a predicate over traces, and has a natural interpretation: the set of valid system traces that it accepts, assuming some value of the intruder's initial knowledge.

3 Confidential and authenticated channels

A confidential channel should protect the confidentiality of any message sent on it from all but the intended recipient. For example, a confidential channel to B can be implemented by encoding the application layer message m as the transport layer message $\{m\}_{PK(B)}$. We identify confidential channels by tagging them with the label C (e.g. writing $C(R_i \rightarrow R_j)$).

The intruder's knowledge is then restricted so that the intruder only learns messages that are sent on non-confidential channels, or that are sent to him. We specify confidential channels by requiring that this definition does indeed capture what the intruder would know after the trace tr. The messages the intruder knows after observing a trace are those that can be deduced from his initial knowledge and the messages sent on the network, so we require that the intruder's knowledge gained from observing all the messages sent on the transport layer is the same as that gained from observing all the application messages, apart from those sent on confidential channels.

We specify authenticated channels by describing the relationship between the *receive* and *send* events performed by the agents at either end of the channel. In particular, we specify under what circumstances an agent may perform a particular *receive* event. The bottom of our hierarchy is the standard Dolev-Yao network model, captured by $\mathcal{N}_1$–$\mathcal{N}_4$.

There are two dishonest events the intruder can perform: faking and hijacking. With some transport protocols the latter can only be performed

using dishonest identities. We specify our channels by placing restrictions on when he can perform these events.

Definition 3.1 (No faking)**.** If $NF(R_i \to R_j)$ then the intruder cannot fake messages on the channel:

$$NF(R_i \to R_j)(tr) \mathrel{\widehat{=}} tr \downarrow \{\mid fake.\hat{R}_i.\hat{R}_j \mid\} = \langle\rangle\,.$$

Definition 3.2 (No-re-ascribing)**.** If $NRA(R_i \to R_j)$ then the intruder cannot change the sender's identity when he hijacks messages:

$$\begin{aligned} NRA(R_i \to R_j)(tr) \mathrel{\widehat{=}} tr \downarrow \{\mid\ & hijack.A \to A'.B \to B' \mid \\ & A, A' : \hat{R}_i; B, B' : \hat{R}_j \cdot A \neq A' \mid\} = \langle\rangle\,. \end{aligned}$$

Definition 3.3 (No-honest-re-ascribing)**.** If $NRA^-(R_i \to R_j)$ then the intruder can only change the sender's identity to a dishonest identity when he hijacks messages:

$$\begin{aligned} NRA^-(R_i \to R_j)(tr) \mathrel{\widehat{=}} tr \downarrow \{\mid\ & hijack.A \to A'.B \to B' \mid \\ & A, A' : \hat{R}_i; B, B' : \hat{R}_j \cdot A \neq A' \wedge Honest(A') \mid\} = \langle\rangle\,. \end{aligned}$$

Definition 3.4 (No-redirecting)**.** If $NR(R_i \to R_j)$ then the intruder cannot redirect messages:

$$\begin{aligned} NR(R_i \to R_j)(tr) \mathrel{\widehat{=}} tr \downarrow \{\mid\ & hijack.A \to A'.B \to B' \mid \\ & A, A' : \hat{R}_i; B, B' : \hat{R}_j \cdot B \neq B' \mid\} = \langle\rangle\,. \end{aligned}$$

Definition 3.5 (No-honest-redirecting)**.** If $NR^-(R_i \to R_j)$ then the intruder cannot redirect messages that were sent to honest agents:

$$\begin{aligned} NR^-(R_i \to R_j)(tr) \mathrel{\widehat{=}} tr \downarrow \{\mid\ & hijack.A \to A'.B \to B' \mid \\ & A, A' : \hat{R}_i; B, B' : \hat{R}_j \cdot B \neq B' \wedge Honest(B) \mid\} = \langle\rangle\,. \end{aligned}$$

These properties are not independent, since no-re-ascribing implies no-honest-re-ascribing, and likewise for no-redirecting. Further, not all combinations are fundamentally different; certain pairs of combinations allow essentially the same intruder behaviours: each simulates the other . We therefore collapse such combinations; for full details of the collapsing cases, see [DL08].

After taking the collapsing cases into consideration we arrive at a hierarchy of four non-confidential and seven confidential channels, shown in Figure 1.

Without taking the collapsing cases into account, there are thirty-six different combinations of the primitives described above; these form a lattice. We describe a point in this lattice by listing each of its components in the order (C, NF, NRA, NR); e.g. the point $(C, \bot, NRA, NR^-)$ is the channel $C \wedge NRA \wedge NR^-$.

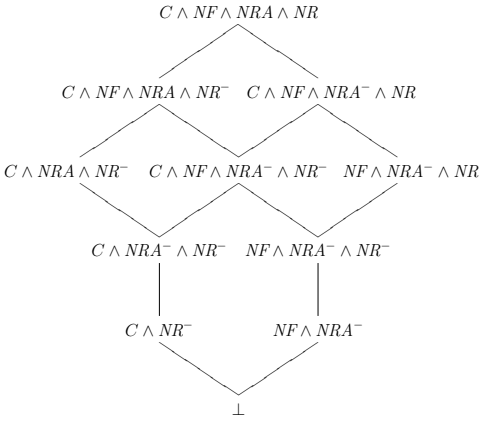


Figure 1: The hierarchy of secure channels.

Definition 3.6. Points in the full lattice are compared component-wise:

$$(c_1, nf_1, nra_1, nr_1) \leqslant (c_2, nf_2, nra_2, nr_2) \Leftrightarrow \\ c_1 \leqslant c_2 \wedge nf_1 \leqslant nf_2 \wedge nra_1 \leqslant nra_2 \wedge nr_1 \leqslant nr_2 \,.$$

The collapsing cases, which are described in detail in [DL08], can also be described by five collapsing rules:

$$\begin{aligned}
\mathcal{C}_1 &\mathrel{\widehat{=}} (\bot,\ \bot\,,\quad x\quad ,\quad y)\ \downarrow\ (\bot,\ \bot\,,\quad \bot\quad ,\quad \bot)\,,\\
\mathcal{C}_2 &\mathrel{\widehat{=}} (x\,,\ NF,\quad \bot\quad ,\quad y)\ \downarrow\ (x,\ \bot\,,\quad \bot\quad ,\quad y)\,,\\
\mathcal{C}_3 &\mathrel{\widehat{=}} (\bot,\ NF,\ NRA,\quad x)\ \downarrow\ (\bot,\ NF,\ NRA^-,\quad x)\,,\\
\mathcal{C}_4 &\mathrel{\widehat{=}} (C,\ \ x\ ,\quad y\quad ,\ \bot)\ \downarrow\ (\bot,\ \ x\ ,\quad y\quad ,\ \bot)\,,\\
\mathcal{C}_5 &\mathrel{\widehat{=}} (C,\ \bot\,,\quad x\quad ,\ NR)\ \downarrow\ (C,\ \bot\,,\quad x\quad ,\ NR^-)\,.
\end{aligned}$$

For example, rule $\mathcal{C}_4$ matches any confidential point that allows redirecting; any channels that match this pattern are collapsed to a non-confidential point with the other components remaining unchanged.

Definition 3.7. For any point (c, nf, nra, nr) in the full lattice, we define $\downarrow (c, nf, nra, nr)$ to be the *collapsed form* of the point. This is the point we reach by continually applying the collapsing rules until we reach a point that cannot be collapsed further.

Proposition 3.8. *The collapsed form of every point in the full lattice is unique and well-defined.*

Proof. With the exception of $\mathcal{C}_2$ and $\mathcal{C}_4$, the patterns for the five rules above are disjoint. The point $(C, NF, \bot, \bot)$ matches the patterns of both $\mathcal{C}_2$ and $\mathcal{C}_4$, so we examine what happens when we apply $\mathcal{C}_2$ first, and when we apply $\mathcal{C}_4$ first:

$$(C, NF, \bot, \bot) \downarrow_{\mathcal{C}_2} (C, \bot, \bot, \bot) \downarrow_{\mathcal{C}_4} (\bot, \bot, \bot, \bot)\,,\\ (C, NF, \bot, \bot) \downarrow_{\mathcal{C}_4} (\bot, NF, \bot, \bot) \downarrow_{\mathcal{C}_2} (\bot, \bot, \bot, \bot)\,.$$

The order in which $\mathcal{C}_2$ and $\mathcal{C}_4$ are applied makes no difference to the resultant collapsed form; in every other case there is at most one rule that can be applied (since the patterns for the rules are disjoint). Therefore the sequence of collapsing rules that can be applied to any given point is unique and well-defined, hence $\downarrow (c, nf, nra, nr)$ is unique and well-defined for any point in the lattice. □

We note that $\downarrow$ is monotonic with respect to the order on the lattice ($\leqslant$).

4 Simulation and alternative specifications

In order to compare the relative strengths of different channels, we need to compare the effect they have on the intruder's capabilities. In particular, we want to check that when the intruder can perform a dishonest activity in two different ways the resulting channels are equivalent. In [DL08] we define a simulation relation that compares channel specifications by comparing the honest agents' views of them. We use this relation to establish an equivalence relation (simulation in both directions) on channel specifications.

The honest agents' view of the traces of a channel specification is the restriction of those traces to the application-layer send and receive events performed by the honest agents. The channel specification $ChannelSpec_1$ simulates $ChannelSpec_2$ if, for all possible values of the intruder's initial knowledge, every trace of the second specification corresponds to a trace of the first specification that appears the same to the honest agents:

$$\forall IIK \subseteq Message \cdot HonestTraces_{IIK}(ChannelSpec_2) \subseteq HonestTraces_{IIK}(ChannelSpec_1)\,.$$

We write $ChannelSpec_1 \preccurlyeq ChannelSpec_2$.

We define our equivalence relation as simulation in both directions; we write $ChannelSpec_1 \cong ChannelSpec_2$. The intruder has exactly the same capabilities in any two equivalent systems: he can perform the same attacks in both, and there is no fact that he can learn in one but not in the other.

Our channels are specified by blocking the dishonest events that the intruder can perform. In [DL08] we give alternative formulations for our channel specifications; these alternatives state exactly which events can occur before an honest agent can receive a message, and require that one of them must occur. We use these forms of the specifications to prove the results in Sections 5 and 6.

We can safely block the intruder's activity when it simulates other activity (that we do not block). For example, we can block the intruder from hijacking his own messages (because he can fake or send to the correct agent in the first place), and from re-ascribing messages to his own identity when they were originally sent to him (because he can learn the message and send it himself).

5 Simple proxies

In this section we present the chaining theorem for *simple proxies*.

Definition 5.1. A *simple proxy* is an agent who is dedicated to forwarding messages from one agent to another. For every pair of roles (R_i, R_j) there is a simple proxy role $Proxy_{(R_i,R_j)}$. For every pair of agents $(A : \hat{R}_i, B : \hat{R}_j)$ such that $A \neq B$ there is a unique simple proxy $P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}$ who forwards messages from A to B. When two roles communicate through a simple proxy, the following trace specification is satisfied:

$$
\begin{array}{l}
SimpleProxies(R_i \to R_j)(tr) \mathrel{\widehat{=}} \\
\quad tr \downarrow \{\!|\, send.\hat{R}_i.Connection.\hat{R}_j \,|\!\} = \langle\rangle \wedge \\
\quad tr \downarrow \{\!|\, receive.\hat{R}_j.Connection.\hat{R}_i \,|\!\} = \langle\rangle \wedge \\
\quad \forall A : \hat{R}_i; B : \hat{R}_j \cdot \exists P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)} \cdot SimpleProxy(P_{(A,B)})(tr) \wedge \\
\quad \forall A, A' : \hat{R}_i; B : \hat{R}_j; P_{(A',B)} : \widehat{Proxy}_{(R_i,R_j)}; c_A : Connection; m : Message_{App} \cdot \\
\qquad send.A.c_A.P_{(A',B)}.m \textbf{ in } tr \wedge Honest(A) \Rightarrow A' = A \wedge \\
\quad \forall A : \hat{R}_i; B, B' : \hat{R}_j; P_{(A,B')} : \widehat{Proxy}_{(R_i,R_j)}; c_B : Connection; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B')}.m \textbf{ in } tr \wedge Honest(B) \Rightarrow B' = B\,,
\end{array}
$$

where each simple proxy satisfies the following specification:

$$
\begin{array}{l}
SimpleProxy(P_{(A,B)})(tr) \mathrel{\widehat{=}} \\
\quad \forall c_P : Connection; m : Message \cdot \\
\qquad \forall A' : Agent \cdot receive.P_{(A,B)}.c_P.A'.m \textbf{ in } tr \Rightarrow A' = A \wedge \\
\qquad \forall B' : Agent \cdot send.P_{(A,B)}.c_P.B'.m \textbf{ in } tr \Rightarrow B' = B \wedge \\
\qquad \exists c'_P : Connection \cdot send.P_{(A,B)}.c_P.B \leqslant receive.P_{(A,B)}.c'_P.A\,.
\end{array}
$$

The simple proxy $P_{(A,B)}$ only establishes connections with A and B, and each connection it establishes is either dedicated to sending messages to B or to receiving messages from A. Further, the simple proxy forwards every message that it receives from A to B.

Because the proxy $P_{(A,B)}$ acts on A's behalf, the proxy is honest if and only if A is honest. We think of the family of proxies $\{P_{(A,B)} \mid B : Agent\}$ as A's proxies (because they all send on her behalf); see Figure 2.

Each simple proxy has a particular job: if $P_{(A,B)}$ receives a message that appears to be from A, he forwards it to B; $P_{(A,B)}$ does not receive messages that appear to have been sent by any other agent. $P_{(A,B)}$ only ever sends messages to B, and never to any other agent. We assume that every agent knows all of its proxies, and also knows which proxies send messages to it,[3] and so if an honest agent who is not B is sent a message from $P_{(A,B)}$ he discards it. We assume that honest agents never attempt to send a message to any simple proxies other than their own.

[3]This could be implemented in the same way as – or even integrated with – a Public Key Infrastructure.

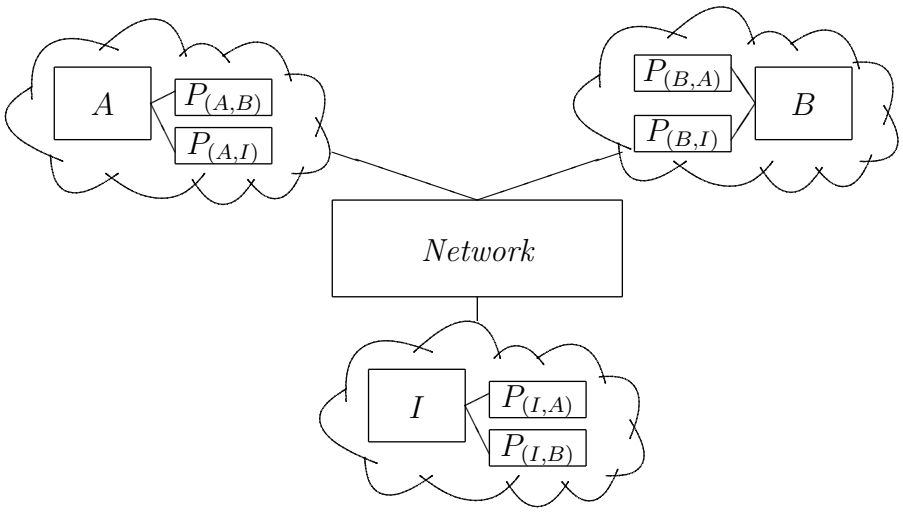


Figure 2: Simple proxies.

5.1 Secure channels through simple proxies

In order to discover which security properties the channel through a simple proxy satisfies we consider each of the components of the hierarchy individually. In the discussion below we refer to the channel to the proxy as $(R_i \rightarrow Proxy_{(R_i,R_j)})$, and the channel from the proxy as $(Proxy_{(R_i,R_j)} \rightarrow R_j)$.

Confidentiality It is clear that if either of the channels to or from a simple proxy is not confidential, the channel through the proxy is not confidential; i.e. the channel through the proxy is confidential only if the channels to and from the proxy are both confidential.

No faking It is also clear that if either of the channels to or from a simple proxy is fakeable, then the channel through the proxy is fakeable. In order to fake a message from A to B, the intruder must either fake sending the message to A's proxy $P_{(A,B)}$, or fake sending the message from A's proxy to B.

No re-ascribing The intruder can either re-ascribe a message on the channel to the proxy or on the channel from the proxy:

1. In order to re-ascribe a message on the channel to the proxy it is not enough for the intruder just to change the sender's identity:

$$tr \mathrel{\widehat{=}} \langle send.A.c_A.P_{(A,B)}.m, hijack.A \rightarrow A'.P_{(A,B)}.c_P.m \rangle .$$

A's proxy will not accept a message that appears to have been sent by another agent (A'). In order to re-ascribe a message on the channel to the proxy, the intruder must also be able to redirect the message to the correct one of the new sender's proxies.

2. On the other hand, re-ascribing a message on the channel from the proxy is straightforward:[4]

$$tr \mathrel{\widehat{=}} \langle \mathit{send}.A.c_A.P_{(A,B)}.m, \mathit{receive}.P_{(A,B)}.c_P.A.m, \\ \mathit{send}.P_{(A,B)}.c'_P.B.m, \mathit{hijack}.P_{(A,B)} \rightarrow P_{(A',B)}.B.c_B.m, \\ \mathit{receive}.B.c_B.P_{(A',B)}.m \rangle .$$

The intruder only needs to change the sender's identity to that of another agent's proxy.

No redirecting The intruder can either redirect a message on the channel to the proxy or on the channel from the proxy:

1. In order to redirect a message on the channel to the proxy the intruder simply redirects the message to a different proxy:

$$tr \mathrel{\widehat{=}} \langle \mathit{send}.A.c_A.P_{(A,B)}.m, \mathit{hijack}.A.P_{(A,B)} \rightarrow P_{(A,B')}.c_P.m, \\ \mathit{receive}.P_{(A,B')}.c_P.A.m, \mathit{send}.P_{(A,B')}.c'_P.B'.m, \\ \mathit{receive}.B'.c_{B'}.P_{(A,B')}.m \rangle .$$

2. On the other hand, in order to redirect a message on the channel from the proxy the intruder cannot just change the recipient's identity:

$$tr \mathrel{\widehat{=}} \langle \mathit{send}.A.c_A.P_{(A,B)}.m, \mathit{receive}.P_{(A,B)}.c_P.A.m \\ \mathit{send}.P_{(A,B)}.c'_P.B.m, \mathit{hijack}.P_{(A,B)}.B \rightarrow B'.c_{B'}.m \rangle .$$

B' will not accept a message from the proxy $P_{(A,B)}$ because B' knows which proxies send messages to him; in order to redirect a message on this channel the intruder must also be able to re-ascribe the message to one of the proxies that B' accepts messages from.

The *SimpleProxies* property on the roles R_i and R_j prevents agents playing role R_i from communicating directly with agents playing role R_j: it insists that they only communicate through proxies. This means that the standard definitions of our secure channels (which restrict the intruder's behaviour when hijacking or faking messages) are vacuously satisfied: there are no messages sent by agents playing role R_i to agents playing role R_j to hijack, and no agent playing role R_j will accept a message that appears to be from an agent playing role R_i.

We have seen that in order to fake a message, the intruder can fake it on the channel to the proxy, or on the channel from the proxy. We have also seen that the intruder can hijack messages on either the channel to the proxy, or on the channel from the proxy. In order to block these activities, we must do so on both channels; we state the definitions of our building blocks on the channel through a simple proxy below.

[4]Note that the proxies $P_{(A,B)}$ and $P_{(A',B)}$ are different agents.

Definition 5.2 (No faking)**.**

$$NF(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} tr \downarrow \{\!|\, fake.\hat{R}_i.\widehat{Proxy}_{(R_i,R_j)},\ fake.\widehat{Proxy}_{(R_i,R_j)}.\hat{R}_j \,|\!\} = \langle\rangle\,.$$

Definition 5.3 (No-re-ascribing)**.**

$$\begin{aligned} &NRA(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\ &\quad tr \downarrow \{\!|\, hijack.A \to A'.P_{(A,B)} \to P_{(A',B')},\ hijack.P_{(A,B)} \to P_{(A',B')}.B \to B' \mid \\ &\qquad A, A' \in \hat{R}_i \wedge B, B' \in \hat{R}_j \wedge P_{(A,B)}, P_{(A',B')} \in \widehat{Proxy}_{(R_i,R_j)} \wedge \\ &\qquad A \neq A' \,|\!\} = \langle\rangle\,. \end{aligned}$$

Definition 5.4 (No-honest-re-ascribing)**.**

$$\begin{aligned} &NRA^{-}(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\ &\quad tr \downarrow \{\!|\, hijack.A \to A'.P_{(A,B)} \to P_{(A',B')},\ hijack.P_{(A,B)} \to P_{(A',B')}.B \to B' \mid \\ &\qquad A, A' \in \hat{R}_i \wedge B, B' \in \hat{R}_j \wedge P_{(A,B)}, P_{(A',B')} \in \widehat{Proxy}_{(R_i,R_j)} \wedge \\ &\qquad A \neq A' \wedge Honest(A') \,|\!\} = \langle\rangle\,. \end{aligned}$$

Definition 5.5 (No-redirecting)**.**

$$\begin{aligned} &NR(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\ &\quad tr \downarrow \{\!|\, hijack.A \to A'.P_{(A,B)} \to P_{(A',B')},\ hijack.P_{(A,B)} \to P_{(A',B')}.B \to B' \mid \\ &\qquad A, A' \in \hat{R}_i \wedge B, B' \in \hat{R}_j \wedge P_{(A,B)}, P_{(A',B')} \in \widehat{Proxy}_{(R_i,R_j)} \wedge \\ &\qquad B \neq B' \,|\!\} = \langle\rangle\,. \end{aligned}$$

Definition 5.6 (No-honest-redirecting)**.**

$$\begin{aligned} &NR^{-}(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\ &\quad tr \downarrow \{\!|\, hijack.A \to A'.P_{(A,B)} \to P_{(A',B')},\ hijack.P_{(A,B)} \to P_{(A',B')}.B \to B' \mid \\ &\qquad A, A' \in \hat{R}_i \wedge B, B' \in \hat{R}_j \wedge P_{(A,B)}, P_{(A',B')} \in \widehat{Proxy}_{(R_i,R_j)} \wedge \\ &\qquad B \neq B' \wedge Honest(B) \,|\!\} = \langle\rangle\,. \end{aligned}$$

In the proof of the simple chaining theorem, below, we show that the alternative specifications for each of the channels in the hierarchy through a simple proxy are satisfied. These forms of the alternative specifications take account of the fact that messages can be faked or hijacked on the channel to the proxy or on the channel from the proxy; the specifications are shown in Appendix A.1. We present one example below; the alternative form of the channel specification $(C \wedge NRA^{-} \wedge NR^{-})(Proxy_{(R_i,R_j)})$ is:

$$
\begin{array}{l}
Alt(C \wedge NRA^{-} \wedge NR^{-})(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy_{(R_i,R_j)}) \wedge C(Proxy_{(R_i,R_j)} \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : Connection \cdot fake.A.P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\quad fake.P_{(A,B)}.B.c_B.m \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m \textbf{ in } tr \vee \ .
\end{array}
$$

5.2 Simple chaining theorem

We make the following observations of the overall channel through a simple proxy.

Observation 5.7. If the intruder cannot redirect messages that were sent to honest agents on the channel to the proxy, then he cannot re-ascribe messages on the channel to the proxy. In order to re-ascribe a message the intruder must be able to redirect the message to one of the new sender's proxies. Further, since all honest agents' proxies are honest, no honest agent ever sends a message to a dishonest agent on the channel to the proxy. Subject to the collapsing cases described earlier, if the channel to the proxy satisfies NR^{-} it also satisfies $NRA \wedge NR$.

Observation 5.8. If the intruder cannot re-ascribe messages to honest agents on the channel from the proxy, then he cannot redirect messages on the channel from the proxy. In order to redirect a message the intruder must be able to re-ascribe it to one of the proxies who sends messages to the new recipient. Subject to the collapsing cases described earlier, if the channel from the proxy satisfies NRA^{-} it also satisfies $NRA^{-} \wedge NR$.

Theorem 5.9 (Simple chaining theorem)**.** *If roles R_i and R_j communicate through simple proxies (i.e. $SimpleProxies(R_i \rightarrow R_j)$) on secure channels such that:*

$$
\begin{array}{l}
ChannelSpec_1(R_i \rightarrow Proxy_{(R_i,R_j)}) \,, \\
ChannelSpec_2(Proxy_{(R_i,R_j)} \rightarrow R_j) \,,
\end{array}
$$

where $ChannelSpec_1$ and $ChannelSpec_2$ are channels in the hierarchy, then the overall channel (through the proxy) satisfies the channel specification:

$$
ChannelSpec =\downarrow (\nwarrow_s ChannelSpec_1 \sqcap \nearrow_s ChannelSpec_2) \,;
$$

where:

$$\begin{aligned}
&\nwarrow_s (c, nf, nra, nr) = \\
&\qquad (c, nf, 2, 2) \ \textit{if}\ nr \geqslant 1\,, \\
&\qquad (c, nf, nra, nr) \ \textit{otherwise}\,; \\
&\nearrow_s (c, nf, nra, nr) = \\
&\qquad (c, nf, nra, 2) \ \textit{if}\ nra \geqslant 1\,, \\
&\qquad (c, nf, nra, nr) \ \textit{otherwise}\,;
\end{aligned}$$

and $\sqcap$ is the greatest lower bound operator in the full lattice.

The proof of the simple chaining theorem is in Section 5.3.

Corollary 5.10. *If roles R_i and R_j communicate through simple proxies (i.e. SimpleProxies$(R_i \to R_j)$) on secure channels such that:*

$$\begin{aligned}
&\mathit{ChannelSpec}(R_i \to \mathit{Proxy}_{(R_i,R_j)})\,, \\
&\mathit{ChannelSpec}(\mathit{Proxy}_{(R_i,R_j)} \to R_j)\,,
\end{aligned}$$

where ChannelSpec is a channel in the hierarchy, then the overall channel (through the proxy) satisfies a channel specification ChannelSpec$'$ such that:

$$\mathit{ChannelSpec} \preccurlyeq \mathit{ChannelSpec}'\,.$$

In particular, ChannelSpec$(\mathit{Proxy}_{(R_i,R_j)})$ holds.

Example 5.11. The channel to the proxy satisfies $C \wedge NRA \wedge NR^-$, and the channel from the proxy satisfies $NF \wedge NRA^- \wedge NR$.

$$\begin{aligned}
&\nwarrow_s (C \wedge NRA \wedge NR^-) = C \wedge NRA \wedge NR\,, \\
&\nearrow_s (NF \wedge NRA^- \wedge NR) = NF \wedge NRA^- \wedge NR\,.
\end{aligned}$$

The greatest lower bound of these two points is $NRA^- \wedge NR$, which collapses to $\bot$.

The channel to the proxy is fakeable and the channel from the proxy is non-confidential; because of collapsing case $\mathcal{C}_1$, the overall channel simulates the bottom channel.

Example 5.12. The channel to the proxy satisfies $NF \wedge NRA^- \wedge NR^-$, and the channel from the proxy satisfies $NF \wedge NRA^-$.

$$\begin{aligned}
&\nwarrow_s (NF \wedge NRA^- \wedge NR^-) = NF \wedge NRA \wedge NR\,, \\
&\nearrow_s (NF \wedge NRA^-) = NF \wedge NRA^- \wedge NR\,.
\end{aligned}$$

The greatest lower bound of these two points is $NF \wedge NRA^- \wedge NR$, which does not collapse. This channel is stronger than both of the individual channels.

The intruder cannot fake messages on this channel, nor can he redirect messages (because he can't redirect messages using the channel to the proxy, and he can't re-ascribe messages using the channel from the proxy). The intruder can only re-ascribe messages with his own identity because this is the greatest capability he has on each channel individually.

Example 5.13. The channel to the proxy satisfies $C \wedge NR^-$, and the channel from the proxy satisfies $C \wedge NRA^- \wedge NR^-$.

$$\begin{aligned} &\nwarrow_s (C \wedge NR^-) = C \wedge NRA \wedge NR\,, \\ &\nearrow_s (C \wedge NRA^- \wedge NR^-) = C \wedge NRA^- \wedge NR\,. \end{aligned}$$

The greatest lower bound of these two points is $C \wedge NRA^- \wedge NR$, which collapses to $C \wedge NRA^- \wedge NR^-$ by $\mathcal{C}_5$. This channel is stronger than the greatest lower bound of the two individual channels.

Although the channel to the proxy is re-ascribable, it only allows the intruder to redirect messages that were sent to him, so the overall channel only allows the intruder to re-ascribe messages to his own identity (on the channel from the proxy).

The resultant channels for all instances of the chaining theorem are shown in Figure 4 (in Appendix B.1).

5.3 An automated proof of the simple chaining theorem

Each instance of Theorem 5.9 is relatively simple to prove. One simply starts with a receive event, and calculates which events are allowed (by the channel specifications to and from the proxy, and by the network rules) to precede it in a valid system trace. Each receive event can be traced back to a set of send, fake, or hijack events; it is then straightforward to determine the strongest channel whose alternative specification is satisfied.

Before we describe the automated proof of the theorem, we show an example proof of one instance.

Lemma 5.14. *If roles R_i and R_j communicate through simple proxies (i.e. SimpleProxies$(R_i \rightarrow R_j)$) on secure channels such that:*

$$\begin{aligned} &(C \wedge NR^-)(R_i \rightarrow Proxy_{(R_i,R_j)})\,, \\ &(C \wedge NRA^- \wedge NR^-)(Proxy_{(R_i,R_j)} \rightarrow R_j)\,, \end{aligned}$$

then $(C \wedge NRA^- \wedge NR^-)(Proxy_{(R_i,R_j)})$.

Proof. We use the *SimpleProxies* property and the alternative specifications of the channels to and from the proxy to show that the alternative form of

$(C \wedge NRA^- \wedge NR^-)(Proxy_{(R_i,R_j)})$ holds; i.e.

$$
\begin{array}{l}
C(R_i \to Proxy_{(R_i,R_j)}) \wedge C(Proxy_{(R_i,R_j)} \to R_j) \wedge \\
\forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\quad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad \exists c_P : Connection \cdot fake.A.P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad fake.P_{(A,B)}.B.c_B.m \textbf{ in } tr \vee \\
\qquad \exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\quad ((A' = A) \vee Dishonest(A)) \wedge ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad hijack.A' \to A.P_{(A',B')} \to P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad hijack.P_{(A',B')} \to P_{(A,B)}.B' \to B.c_B.m \textbf{ in } tr \vee \ .
\end{array}
\tag{$\dagger$}
$$

holds for all valid system traces tr such that for all prefixes $tr' \leqslant tr$:

$$
\begin{array}{l}
SimpleProxies(R_i \to R_j)(tr') \,, \\
(C \wedge NR^-)(R_i \to Proxy_{(R_i,R_j)})(tr') \,, \\
(C \wedge NRA^- \wedge NR^-)(Proxy_{(R_i,R_j)} \to R_j)(tr') \,.
\end{array}
$$

It is clear that $C(R_i \to Proxy_{(R_i,R_j)})(tr)$ and $C(Proxy_{(R_i,R_j)} \to R_j)(tr)$ both hold, so we must show that the second half of ($\dagger$) holds.

Let A and B be agents playing roles R_i and R_j, $P_{(A,B)}$ be A's proxy to B, c_B a connection and m an application-layer message. Suppose that the event $receive.B.c_B.P_{(A,B)}.m$ occurs in tr; the network rule $\mathcal{N}_4$ implies the existence of one of several events earlier in the trace. The set of possible events is limited by $C \wedge NRA^- \wedge NR^-$, which holds on the channel from the proxy:

1. $\exists c_P : Connection \cdot send.P_{(A,B)}.c_P.B.m \textbf{ in } tr$;
2. $fake.P_{(A,B)}.B.c_B.m \textbf{ in } tr$;
3. $\exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)} \cdot$
   $hijack.P_{(A',B')} \to P_{(A,B)}.B' \to B.c_B.m \textbf{ in } tr \wedge$
   $((P_{(A',B')} = P_{(A,B)}) \vee Dishonest(P_{(A,B)})) \wedge$
   $((B' = B) \vee Dishonest(B'))$.

We consider each of these possibilities independently because each one leads to a different trace.

1. Suppose that $send.P_{(A,B)}.c_P.B.m$ precedes the receive event in the trace tr, for some connection identifier c_P. $SimpleProxies(R_i \to R_j)$ implies $SimpleProxy(P_{(A,B)})(tr)$, and so $receive.P_{(A,B)}.c'_P.A.m$ occurs earlier in the trace, for some connection c'_P.

   We use network rule $\mathcal{N}_4$ again; this implies the existence of one of several events earlier in the trace, and, as before, these possibilities are limited by $C \wedge NR^-$, which holds on the channel to the proxy:

(a) $\exists c_A : \mathit{Connection} \cdot \mathit{send}.A.c_A.P_{(A,B)}.m$ **in** tr ;

(b) $\mathit{fake}.A.P_{(A,B)}.c'_P.m$ **in** tr ;

(c) $\exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)} \cdot$
$\mathit{hijack}.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c'_P.m$ **in** $tr \wedge$
$((P_{(A',B')} = P_{(A,B)}) \vee \mathit{Dishonest}(P_{(A',B')}))$.

The first two of these disjuncts match two of the disjuncts in (†), so we do not (and cannot) trace these back further.

In the third disjunct, if $A' \neq A$ then $P_{(A',B')} \neq P_{(A,B)}$, so $P_{(A',B')}$ is dishonest, and hence A' is dishonest. A must be honest (because the intruder does not re-ascribe his own messages to himself), so this trace simulates one in which the intruder fakes the message with A's identity.

If $B' \neq B$ then the same argument shows that A' is dishonest; again, if $A' \neq A$ then this trace simulates one in which the intruder fakes the message; if $A' = A$ then the intruder has redirected his own message: this simulates a trace in which the intruder sends the message to the correct agent in the first place.

Once we discount the simulating traces, we conclude that $A' = A$ and $B' = B$. This disjunct is now more restrictive than the corresponding disjunct in (†).

2. The second possibility is that the intruder faked the message with the proxy's identity; this disjunct is already in the correct form for (†).

3. The final possibility is that the intruder hijacked a message sent by the simple proxy $P_{(A',B')}$ to the agent B'. If $P_{(A',B')} = P_{(A,B)}$, then necessarily $A' = A$; if $P_{(A,B)}$ is dishonest, A is also dishonest. Since we already know that either $B' = B$, or B' is dishonest, this disjunct matches that in (†).

We have shown that if an agent B receives a message from A's proxy $P_{(A,B)}$, then the set of events that may precede this receive event is a subset of those allowed by the alternative form of the proxy channel specification $C \wedge NRA^- \wedge NR^-$ on the channel $R_i \rightarrow R_j$. Therefore, $(C \wedge NRA^- \wedge NR^-)(Proxy_{(R_i,R_j)})$ holds. □

While each instance of the theorem can be proved simply, there are 121 instances[5] that must be proved. In order to ease this process we have developed a Haskell[6] script that performs the proofs automatically; the script listing is shown in Appendix C. In the rest of this section we describe the script, and relate its various stages to the proof example shown above.

[5]There are 11 possibilities for the channel to the proxy, and 11 for the channel from the proxy.

[6]See http://www.haskell.org/

Deriving the full set of trace patterns We first calculate the distinct trace patterns that result in an honest agent receiving a message, via a proxy, from another honest agent or the intruder. A trace pattern is a subtrace consisting of the events leading up to a receive event in which all identities, connection identifiers and message values are representative. For example, a trace pattern may show that an honest agent sends a message to their proxy, the proxy receives it and then sends it on, the intruder then redirects the message to another honest agent, and then the new recipient receives the message; e.g.

$$s \mathrel{\widehat{=}} \langle \mathit{send}.A.c_A.P_{(A,B)},\ \mathit{receive}.P_{(A,B)}.c_P.A.m,\ \mathit{send}.P_{(A,B)}.c'_P.B.m,\\ \mathit{hijack}.P_{(A,B)} \rightarrow P_{(A,B')}.B \rightarrow B'.c_{B'}.m,\ \mathit{receive}.B'.c_{B'}.P_{(A,B')}.m \rangle\ .$$

Applying the channel properties We apply the properties of the channels to and from the proxy to eliminate those trace patterns in which the intruder must perform an event that the channel does not allow him to. For example, if the intruder cannot fake on the channel to the proxy, we eliminate those trace patterns in which he fakes a message on this channel.

Determining the resultant channel specification We examine the remaining trace patterns to determine which capabilities the intruder still has. For example, if one of the remaining trace patterns shows that an honest agent A sent a message to an honest agent B, but then B receives that message from the dishonest agent I, then this pattern demonstrates that the intruder can re-ascribe messages with his own identity. When we examine each of the trace patterns we discover which events the intruder can perform; we then find the point in the lattice that corresponds to these remaining events, and collapse this point to a channel in the hierarchy.

We describe each of these three stages in more detail below.

5.3.1 Deriving the full set of trace patterns

To derive the full set of trace patterns, we do not assume that either the channel to the proxy or the channel from the proxy satisfy any secure channel specifications. Suppose that tr is a valid system trace such that for all prefixes $tr' \leqslant tr$, $\mathit{SimpleProxies}(R_i \rightarrow R_j)(tr')$ holds.

Let A and B be agents playing roles $\hat{R}_i$ and $\hat{R}_j$, $P_{(A,B)}$ be A's proxy to B, c_B a connection and m an application-layer message. Suppose that the event $\mathit{receive}.B.c_B.P_{(A,B)}.m$ occurs in tr; the network rule $\mathcal{N}_4$ implies the existence of one of several events earlier in the trace.

1. $\mathit{send}.P_{(A,B)}.c_P.B.m$ **in** tr for some connection c_P;

2. $\mathit{fake}.P_{(A,B)}.B.c_B.m$ **in** tr ;

3. $\exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)} \cdot$
   $\mathit{hijack}.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m$ **in** tr .

There are three different possibilities: either the proxy sent the message to B, the intruder faked the message to B (with the proxy's identity), or the intruder hijacked a message sent by a different proxy ($P_{(A',B')}$). Each of these events leads to different trace patterns, so we investigate them independently.

1. The event $\mathit{send}.P_{(A,B)}.c_P.B.m$ occurs in the trace for some connection c_P. Since B accepts this message, $\mathit{SimpleProxies}(R_i \rightarrow R_j)$ implies that $\mathit{SimpleProxy}(P_{(A,B)})$ holds for tr, and so $P_{(A,B)}$ must previously have received this message. The event $\mathit{receive}.P_{(A,B)}.c'_P.A.m$ occurs earlier in the trace, for some connection c'_P.

   We apply network rule $\mathcal{N}_4$ again; this implies the existence of one of several events earlier in the trace.

   (a) $\mathit{send}.A.c_A.P_{(A,B)}$ **in** tr for some connection c_A; the trace has the following pattern:

   $$tr_1 \mathrel{\widehat{=}} \langle \mathit{send}.A.c_A.P_{(A,B)}.m,\ \mathit{receive}.P_{(A,B)}.c'_P.A.m,\ \mathit{send}.P_{(A,B)}.c_P.B.m,\ \mathit{receive}.B.c_B.P_{(A,B)}.m \rangle .$$

   (b) $\mathit{fake}.A.P_{(A,B)}.c'_P.m$ **in** tr ; the trace has the following pattern:

   $$tr_2 \mathrel{\widehat{=}} \langle \mathit{fake}.A.P_{(A,B)}.c'_P.m,\ \mathit{receive}.P_{(A,B)}.c'_P.A.m,\ \mathit{send}.P_{(A,B)}.c_P.B.m,\ \mathit{receive}.B.c_B.P_{(A,B)}.m \rangle .$$

   The intruder does not fake messages with his own identity, so in this trace pattern we assume that A is honest.

   (c) $\exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)} \cdot$
   $\mathit{hijack}.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c'_P.m$ **in** tr .
   The intruder can only hijack messages that were previously sent ($\mathcal{N}_2$), so A' must have sent the message to her proxy earlier in the trace (from some connection $c_{A'}$). The trace has the following pattern:

   $$tr_3 \mathrel{\widehat{=}} \langle \mathit{send}.A'.c'_A.P_{(A',B')}.m,\ \mathit{hijack}.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c'_P.m,\ \mathit{receive}.P_{(A,B)}.c'_P.A.m,\ \mathit{send}.P_{(A,B)}.c_P.B.m,\ \mathit{receive}.B.c_B.P_{(A,B)}.m \rangle .$$

   We can safely block the intruder from hijacking his own messages, so in this trace pattern we assume that A' is honest. If $A' = A$

and $B' = B$, then this trace pattern just shows a replay on the channel to the proxy; because none of our channels prevent replays, we are not interested in whether or not the intruder has this capability. We assume that either $A' \neq A$ or $B' \neq B$.

2. The event $\mathit{fake}.P_{(A,B)}.B.c_B$ occurs in the trace. Since the intruder does not fake with dishonest identities, and since the proxy $P_{(A,B)}$ is honest if and only if A is honest, we assume that A is honest. The trace has the following pattern:

$$tr_4 \mathrel{\widehat{=}} \langle \mathit{fake}.P_{(A,B)}.B.c_B.m,\ \mathit{receive}.B.c_B.P_{(A,B)}.m \rangle\,.$$

3. The event $\mathit{hijack}.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m$ occurs in the trace. We apply $\mathcal{N}_2$ again: the proxy $P_{(A',B')}$ must have sent the message to B' earlier in the trace (in some connection $c_{P'}$). So far, the trace has this pattern:

$$\begin{aligned} s \mathrel{\widehat{=}} \langle & \mathit{send}.P_{(A',B')}.c_{P'}.B'.m, \\ & \mathit{hijack}.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m,\ \mathit{receive}.B.c_B.P_{(A,B)}.m \rangle\,. \end{aligned}$$

As before, we assume that $P_{(A',B')}$ is honest (and hence A' is honest). Now $\mathit{SimpleProxies}(R_i \rightarrow R_j)$ applies again and implies that $\mathit{SimpleProxy}(P_{(A',B')})$ holds for tr, and so $P_{(A',B')}$ must previously have received this message from A' (in some connection $c'_{P'}$). The trace now has the following pattern:

$$\begin{aligned} s \mathrel{\widehat{=}} \langle & \mathit{receive}.P_{(A',B')}.c'_{P'}.A',\ \mathit{send}.P_{(A',B')}.c_{P'}.B'.m, \\ & \mathit{hijack}.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m,\ \mathit{receive}.B.c_B.P_{(A,B)}.m \rangle\,. \end{aligned}$$

We apply network rule $\mathcal{N}_4$ again; this implies the existence of one of several events earlier in the trace.

   (a) $\mathit{send}.A'.c_{A'}.P_{(A',B')}.m$ **in** tr for some connection $c_{A'}$; the trace has the following pattern:

$$\begin{aligned} tr_5 \mathrel{\widehat{=}} \langle & \mathit{send}.A'.c_{A'}.P_{(A',B')}.m, \\ & \mathit{receive}.P_{(A',B')}.c'_{P'}.A',\ \mathit{send}.P_{(A',B')}.c_{P'}.B'.m, \\ & \mathit{hijack}.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m,\ \mathit{receive}.B.c_B.P_{(A,B)}.m \rangle\,. \end{aligned}$$

   As before, we assume that A' is honest and that either $A' \neq A$ or $B' \neq B$.

   (b) $\mathit{fake}.A'.P_{(A',B')}.c'_{P'}.m$ **in** tr. If the intruder can fake with A''s identity, he can also fake with A's identity; this means that the hijack event on the channel from the proxy is unnecessary. We ignore this trace pattern.

(c) $\exists A'' : \hat{R}_i; B'' : \hat{R}_j; P_{(A'',B'')} : \widehat{Proxy}_{(R_i,R_j)} \cdot$
$hijack.A'' \rightarrow A'.P_{(A'',B'')} \rightarrow P_{(A',B')}.c'_{P'}.m$ **in** tr .

We apply $\mathcal{N}_2$ again: the agent A'' must have sent the message to her proxy in order for the intruder to hijack it. The trace now has the following pattern:

$$\begin{aligned} tr_6 \mathrel{\widehat{=}} \langle & send.A''.c_{A''}.P_{(A'',B'')}.m, \\ & hijack.A'' \rightarrow A'.P_{(A'',B'')} \rightarrow P_{(A',B')}.c'_{P'}.m, \\ & receive.P_{(A',B')}.c'_{P'}.A',\ send.P_{(A',B')}.c_{P'}.B'.m, \\ & hijack.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m, \\ & receive.B.c_B.P_{(A,B)}.m \rangle \ . \end{aligned}$$

If $A'' \neq A'$ and $A' \neq A$ then the intruder has re-ascribed the message to A' on the channel to the proxy; he then re-ascribes it to A on the channel from the proxy. This is unnecessary activity, and this trace pattern simulates one in which he doesn't hijack on the channel to the proxy. We therefore assume that either $A'' = A'$ or $A' = A$; similarly we assume that either $B'' = B'$ or $B' = B$. We ignore the trace patterns in which $A'' = A'$ and $B'' = B'$, or $A' = A$ and $B' = B$); we assume that A'' is honest.

5.3.2 Applying the channel properties

The Haskell script automatically generates all of the trace patterns described in the previous section. It then takes each of the 121 combinations of channels to and from the proxy and uses the channel specifications to eliminate those traces that are not allowed. The traces are calculated in two stages: the set of activity on the channel from the proxy is calculated first, and then the channel specification on the channel from the proxy is applied; then the activity on the channel to the proxy is calculated, and finally the channel specification on the channel to the proxy is applied. The result is a set of trace patterns $\{tr_1, tr_2, \ldots, tr_n\}$ that are allowed on the resultant channel.

5.3.3 Determining the resultant channel specification

In order to determine which channel property the resultant channel satisfies we examine each of the co-ordinate points in the lattice individually. The resultant channel is confidential (the first co-ordinate point) if and only if the channels to and from the proxy are confidential; we call this value c. We determine the remaining three co-ordinate points (no-faking, no-re-ascribing and no-redirecting) by looking at each of the allowed trace patterns.

Each of these trace patterns demonstrates that the intruder can perform a particular event. We present below a summary of these patterns; each case shows the initial and final events in the pattern and then describes

which activity this demonstrates. This mapping allows us to assign a tuple (nf, nra, nr) to each trace pattern.

When the final event is $receive.B.c_B.P_{(A,B)}.m$ (agent B receives a message from the honest agent A's proxy):

$\boldsymbol{send}.A.c_A.P_{(A,B)}.m$ this does not demonstrate any activity: $(1, 2, 2)$;

$\boldsymbol{send}.A.c_A.P_{(A,I)}.m$ the intruder redirected a message that was sent to a dishonest agent: $(1, 2, 1)$;

$\boldsymbol{send}.A.c_A.P_{(A,B')}.m$ the intruder redirected a message that was sent to an honest agent: $(1, 2, 0)$;

$\boldsymbol{send}.A'.c_{A'}.P_{(A',B)}.m$ the intruder re-ascribed a message to an honest agent: $(1, 0, 2)$;

$\boldsymbol{send}.A'.c_{A'}.P_{(A',I)}.m$ the intruder re-ascribed a message to an honest agent, and redirected a message sent to a dishonest agent: $(1, 0, 1)$;

$\boldsymbol{send}.A'.c_{A'}.P_{(A',B')}.m$ the intruder re-ascribed a message to an honest agent, and redirected a message that was sent to an honest agent: $(1, 0, 0)$;

$\boldsymbol{send}.I.c_I.P_{(I,B)}.m$ the intruder hijacked his own message to simulate a fake: $(0, 2, 2)$;

$\boldsymbol{fake}.A.P_{(A,B)}.c_P.m$ the intruder faked a message to the proxy: $(0, 2, 2)$;

$\boldsymbol{fake}.P_{(A,B)}.B.c_B.m$ the intruder faked a message from the proxy: $(0, 2, 2)$.

When the final event is $receive.B.c_B.P_{(I,B)}.m$ (agent B receives a message from the dishonest agent I's proxy):

$\boldsymbol{send}.I.c_I.P_{(I,B)}.m$ this does not demonstrate any activity: $(1, 2, 2)$;

$\boldsymbol{send}.A.c_A.P_{(A,B)}.m$ the intruder re-ascribed a message to a dishonest agent: $(1, 1, 2)$;

$\boldsymbol{send}.A.c_A.P_{(A,I)}.m$ this does not demonstrate any activity (the intruder sent a message that was sent to him): $(1, 2, 2)$;

$\boldsymbol{send}.A.c_A.P_{(A,B')}.m$ the intruder re-ascribed a message to a dishonest agent, and redirected a message sent to an honest agent; however, this is only possible on a non-confidential channel, so this simulates a learn and send: $(1, 2, 2)$.

The result of applying the mapping to each of the trace patterns is a set of tuples of the form (nf_i, nra_i, nr_i) for $i = 1 \ldots n$. We take the value of

the confidential co-ordinate and calculate the lattice point of the resultant channel in the following way:

$$(c, nf, nra, nr) = (c, \mathbf{min}_{i=1}^{n}(nf_i), \mathbf{min}_{i=1}^{n}(nra_i), \mathbf{min}_{i=1}^{n}(nr_i)) .$$

Finally we collapse this point to a point in the channel hierarchy. Thus the resultant channel satisfies the channel specification:

$$(c, nf, nra, nr) =\downarrow (c, \mathbf{min}_{i=1}^{n}(nf_i), \mathbf{min}_{i=1}^{n}(nra_i), \mathbf{min}_{i=1}^{n}(nr_i)) .$$

By eliminating trace patterns and calculating the resultant point in the hierarchy in this manner we prove that the alternative specification of the resultant channel holds on all valid system traces in which the channel specifications to and from the proxy hold.

6 Multiplexing proxies

In this section we consider the more general (multiplexing) proxy case. The study of simple proxies shows that by chaining two secure channels through a trusted third party one can sometimes produce a stronger channel. However, in the simple case, we thought of the proxies as 'belonging' to one of the agents communicating; it is highly likely that A trusts her proxies, but should she trust other agents' proxies who send messages to her? In this section we consider more general multiplexing proxies. A multiplexing proxy is a trusted third party who is willing to forward messages from any agent to any other agent.

We assume that all multiplexing proxies are honest. There is nothing to stop the intruder from setting up proxies of his own; however, any message sent through a dishonest proxy cannot remain confidential, and any message received from a dishonest proxy cannot be authenticated.

When agent A intends to send a message to another agent (B) through a simple proxy she just needs to pick the correct simple proxy to send the message to. The proxy knows whom to forward the message to because it is dedicated to that job. If A is to use a multiplexing proxy, she must communicate her intent (to talk to B) to the proxy. Similarly, when B receives a message from A's proxy, he knows who originally sent the message; when B receives a message from a multiplexing proxy, there must be some communication from the proxy to B to say whom the message is from.

One way to solve this problem would be to build a special transport-layer protocol in which the message sender's protocol agent tells the proxy whom to establish a connection with. The agent may then just send messages to the proxy (just as they would if they were sending the messages directly to the recipient). Similarly, the proxy tells the recipient's protocol agent whom the messages are from. However, this solution is unsuitable for our model for two reasons:

- The whole point of the model is to make the details of the transport-layer protocol abstract; once we start to impose conditions on the transport-layer protocol, we lose the generality of the abstract model;
- When we discussed simple proxies we argued the case for considering the proxies as application-layer entities; we make the same argument here as we wish all details of the discussion to be at the application-layer.

The solution we adopt, therefore, is to annotate the application-layer messages with information about whom they are intended for, and whom they were originally sent by. In order to send a message m to B (via the multiplexing proxy P), agent A concatenates B's identity to the message:

$$\mathit{send}.A.c_A.P.\langle m, B\rangle\,.$$

When P receives this message he concatenates it to A's identity, and sends it on to B:

$$\mathit{send}.P.c_P.B.\langle A, m\rangle\,.$$

This only works if the channel is either confidential or non-fakeable; however, all of our channels satisfy at least one of these two properties, so this method can be used on all of our channels.

We assume that none of the application-layer protocols call for agents to send messages with the same type as the messages described above. If we do not make this assumption, it might be possible for messages created by honest agents for use in the application-layer protocols to be mistaken for messages sent to or from a multiplexing proxy.

Definition 6.1. A *multiplexing proxy* is an honest agent who is dedicated to forwarding messages; there is a single proxy role *Proxy*. When two roles communicate through a multiplexing proxy, the following trace specification is satisfied:

$$\begin{aligned}
&\mathit{Proxies}(R_i \to R_j)(tr) \mathrel{\widehat{=}} \\
&\quad tr \downarrow \{\!|\ \mathit{send}.\hat{R}_i.\mathit{Connection}.\hat{R}_j\ |\!\} = \langle\rangle \wedge \\
&\quad tr \downarrow \{\!|\ \mathit{receive}.\hat{R}_j.\mathit{Connection}.\hat{R}_i\ |\!\} = \langle\rangle \wedge \\
&\quad \forall A, B, P : \mathit{Agent}; c_A : \mathit{Connection}; m : \mathit{Message} \cdot \\
&\qquad \mathit{send}.A.c_A.P.\langle m, B\rangle \ \mathbf{in}\ tr \Rightarrow \mathit{Proxy}(P)(tr) \wedge \\
&\quad \forall A, B, P : \mathit{Agent}; c_B : \mathit{Connection}; m : \mathit{Message} \cdot \\
&\qquad \mathit{receive}.B.c_B.P.\langle A, m\rangle \ \mathbf{in}\ tr \Rightarrow \mathit{Proxy}(P)(tr)\,,
\end{aligned}$$

where each multiplexing proxy satisfies the following specification:

$$
\begin{array}{l}
\mathit{Proxy}(P)(tr) \mathrel{\widehat{=}} \\
\quad \mathit{Honest}(P) \wedge \\
\quad \forall c_P : \mathit{Connection}; A, B, B' : \mathit{Agent}; m, m' : \mathit{Message} \cdot \\
\qquad \mathit{receive}.P.c_P.A.\langle m, B\rangle \ \mathbf{in}\ tr \wedge \mathit{receive}.P.c_P.A.\langle m', B'\rangle \ \mathbf{in}\ tr \Rightarrow B = B' \wedge \\
\quad \forall c_P : \mathit{Connection}; A, A', B : \mathit{Agent}; m, m' : \mathit{Message} \cdot \\
\qquad \mathit{send}.P.c_P.B.\langle A, m\rangle \ \mathbf{in}\ tr \wedge \mathit{send}.P.c_P.B.\langle A', m'\rangle \ \mathbf{in}\ tr \Rightarrow A = A' \wedge \\
\quad \forall c_P : \mathit{Connection}; A : \mathit{Agent} \cdot \exists c'_P : \mathit{Connection}; B : \mathit{Agent} \cdot \\
\qquad \mathit{send}.P.c_P.\langle A, m\rangle \leqslant \mathit{receive}.P.c'_P.\langle m, B\rangle \,.
\end{array}
$$

Each connection that the multiplexing proxies establish is either dedicated to receiving messages from one agent, or sending messages to one agent. Although two individual messages from one agent to another could be sent through different proxies, we assume that all the messages in one connection are sent to (or received from) the same proxy.

We assume that the honest agents only send messages of the form $\langle m, B\rangle$ to proxies, and that they will only receive messages of the form $\langle A, m\rangle$ from proxies.

Each multiplexing proxy can be used by several agents. One can imagine a scenario in which each organisation has a pool of multiplexing proxies: every agent in that organisation communicates with external agents through the proxies, but communicates directly with internal agents (see Figure 3).

Whenever a multiplexing proxy receives a message from A for B, he forwards it to B, and tells B that it is from A. In each connection, the proxies only exchange messages with one other agent; they also only allow one third party to be involved in each connection; this is so that, later, we can show that the session properties are invariant under chaining.

6.1 Secure channels through multiplexing proxies

The public knowledge of the role of each simple proxy was what lead to the rather surprising result that the chained form of two channels can be stronger than both channels individually. With the multiplexing proxies we no longer have this public knowledge; B only knows whom the message was originally sent by by examining it and seeing whose identity is attached to it. As we did last time, we consider each of the components of the hierarchy individually in order to discover which properties the channel through a proxy satisfies. In the discussion below we refer to the channel to the proxy as $(R_i \rightarrow \mathit{Proxy})$ and the channel from the proxy as $(\mathit{Proxy} \rightarrow R_j)$.

Confidentiality If either of the channels to or from a proxy is not confidential, then the channel through the proxy is not confidential. Since all multiplexing proxies are honest, the channel through the proxy

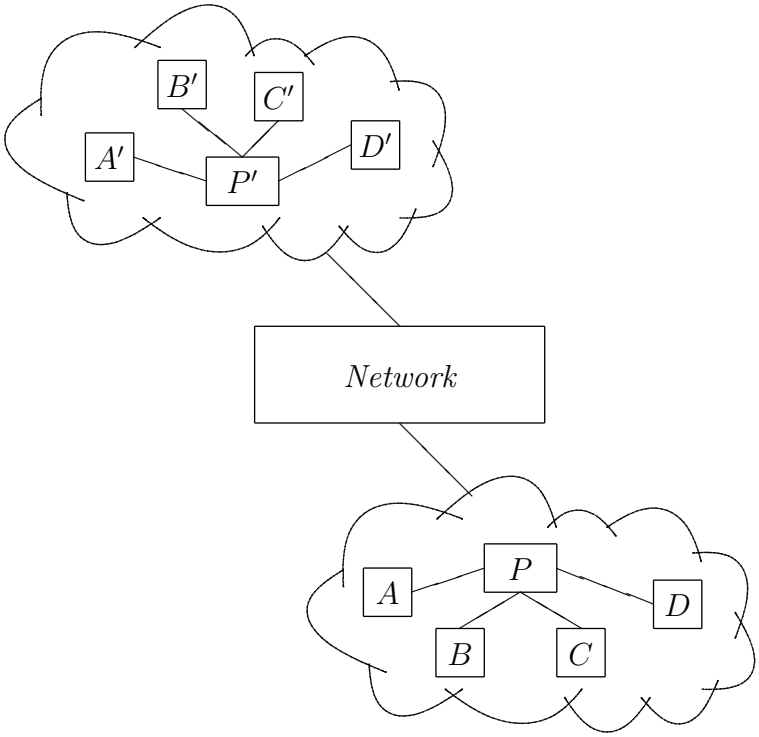


Figure 3: Multiplexing proxies.

is confidential if and only if the channels to and from the proxy are confidential.

No faking It is also clear that if either of the channels to or from a proxy is fakeable, then the channel through the proxy is fakeable. In order to fake a message from A to B, the intruder must either fake sending the message to the proxy, or from the proxy.

No re-ascribing Unlike the simple proxies, the intruder cannot choose which channel to re-ascribe a message on: he must do so on the channel to the proxy. This is straightforward:

$$tr \mathrel{\widehat{\in}} \langle send.A.c_A.P.\langle m, B\rangle, hijack.A \rightarrow A'.P.c_P.\langle m, B\rangle\rangle \,.$$

The only identity that the intruder can change by re-ascribing on the channel from the proxy is that of the message sender (the proxy):

$$\begin{aligned} tr \mathrel{\widehat{\in}} \langle & send.A.c_A.P.\langle m, B\rangle, receive.P.c_P.A.\langle m, B\rangle, \\ & send.P.c'_P.B.\langle A, m\rangle, hijack.P \rightarrow P'.B.c_B.\langle A, m\rangle \,. \end{aligned}$$

Because honest agents only accept messages of the form $\langle A, m\rangle$ from proxies, the intruder can only re-ascribe the message to a different proxy: he cannot change the identity of the original sender of the message by re-ascribing the message on the channel from the proxy.

No redirecting The intruder can only redirect a message using the channel from the proxy; this is straightforward:

$$\begin{aligned} tr \mathrel{\widehat{\in}} \langle & send.A.c_A.P.\langle m, B\rangle, receive.P.c_P.A.\langle m, B\rangle, \\ & send.P.c'_P.B.\langle A, m\rangle, hijack.P.B \rightarrow B'.c_{B'}.\langle A, m\rangle\rangle \,. \end{aligned}$$

The only identity that the intruder can change by redirecting the message on the channel to the proxy is that of the message recipient: in this case, the proxy;

$$tr \mathrel{\widehat{\in}} \langle send.A.c_A.P.\langle m, B\rangle, hijack.A.P \rightarrow P'.c_{P'}.\langle m, B\rangle\rangle \,.$$

Because the only honest agents who receive messages of the form $\langle m, B\rangle$ are proxies, the intruder can only redirect the message to a different proxy.

The *Proxies* property on the roles R_i and R_j prevents agents playing role R_i from communicating directly with agents playing role R_j. As before, we must reframe the definitions of the authenticated channel building blocks for the channel through a multiplexing proxy because the standard definitions are vacuously satisfied.

Definition 6.2 (No faking)**.**

$$
\begin{aligned}
&NF(Proxy(R_i \to R_j))(tr) \mathrel{\widehat{=}} \\
&\quad tr \restriction \{| \mathit{fake}.A.P.c_A.\langle m, B\rangle, \mathit{fake}.P.c_P.B.\langle A, m\rangle \mid \\
&\qquad\qquad A \in \hat{R}_i \wedge P \in \widehat{Proxy} \wedge B \in \hat{R}_j \wedge \\
&\qquad\qquad c_A, c_P \in \mathit{Connection} \wedge m \in \mathit{Message}_{App} \mid\} = \langle\rangle \,.
\end{aligned}
$$

Definition 6.3 (No-re-ascribing)**.**

$$
\begin{aligned}
&NRA(Proxy(R_i \to R_j))(tr) \mathrel{\widehat{=}} \\
&\quad tr \restriction \{| \mathit{hijack}.A \to A'.P \to P'.c_{P'}.\langle m, B\rangle \mid \\
&\qquad\qquad A, A' \in \hat{R}_i \wedge P, P' \in \widehat{Proxy} \wedge B \in \hat{R}_j \wedge c_{P'} \in \mathit{Connection} \wedge \\
&\qquad\qquad m \in \mathit{Message}_{App} \wedge A \neq A' \mid\} = \langle\rangle \,.
\end{aligned}
$$

Definition 6.4 (No-honest-re-ascribing)**.**

$$
\begin{aligned}
&NRA^{-}(Proxy(R_i \to R_j))(tr) \mathrel{\widehat{=}} \\
&\quad tr \restriction \{| \mathit{hijack}.A \to A'.P \to P'.c_{P'}.\langle m, B\rangle \mid \\
&\qquad\qquad A, A' \in \hat{R}_i \wedge P, P' \in \widehat{Proxy} \wedge B \in \hat{R}_j \wedge c_{P'} \in \mathit{Connection} \wedge \\
&\qquad\qquad m \in \mathit{Message}_{App} \wedge A \neq A' \wedge \mathit{Honest}(A') \mid\} = \langle\rangle \,.
\end{aligned}
$$

Definition 6.5 (No-redirecting)**.**

$$
\begin{aligned}
&NR(Proxy(R_i \to R_j))(tr) \mathrel{\widehat{=}} \\
&\quad tr \restriction \{| \mathit{hijack}.P \to P'.B \to B'.c_{B'}.\langle A, m\rangle \mid \\
&\qquad\qquad A \in \hat{R}_i \wedge P, P' \in \widehat{Proxy} \wedge B, B' \in \hat{R}_j \wedge c_{B'} \in \mathit{Connection} \wedge \\
&\qquad\qquad m \in \mathit{Message}_{App} \wedge B \neq B' \mid\} = \langle\rangle \,.
\end{aligned}
$$

Definition 6.6 (No-honest-redirecting)**.**

$$
\begin{aligned}
&NR^{-}(Proxy(R_i \to R_j))(tr) \mathrel{\widehat{=}} \\
&\quad tr \restriction \{| \mathit{hijack}.P \to P'.B \to B'.c_{B'}.\langle A, m\rangle \mid \\
&\qquad\qquad A \in \hat{R}_i \wedge P, P' \in \widehat{Proxy} \wedge B, B' \in \hat{R}_j \wedge c_{B'} \in \mathit{Connection} \wedge \\
&\qquad\qquad m \in \mathit{Message}_{App} \wedge B \neq B' \wedge \mathit{Honest}(B) \mid\} = \langle\rangle \,.
\end{aligned}
$$

As in the proof of the simple chaining theorem, in the proof of the chaining theorem, below, we show that the alternative specifications for each of the channels in the hierarchy through a proxy are satisfied. These forms of the alternative specifications take account of the fact that messages can be faked or hijacked on the channel to the proxy or on the channel from the proxy; the specifications are shown in Appendix B.2. We present one example below; the alternative form of the channel specification $(C \wedge NRA^{-} \wedge NR^{-})(Proxy(R_i \to R_j))$ is:

$$
\begin{array}{l}
\mathit{Alt}(C \wedge \mathit{NRA}^- \wedge \mathit{NR}^-)(\mathit{Proxy}(R_i \to R_j))(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \to \mathit{Proxy}) \wedge C(\mathit{Proxy} \to R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : \mathit{Connection}; A : \hat{R}_i; P : \widehat{\mathit{Proxy}}; m : \mathit{Message}_{\mathit{App}} \cdot \\
\qquad \mathit{receive}.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : \mathit{Connection} \cdot \mathit{send}.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : \mathit{Connection} \cdot \mathit{fake}.A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \mathit{fake}.P.B.c_B.\langle A, m\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists P' : \widehat{\mathit{Proxy}}; B' : \hat{R}_j; c_B : \mathit{Connection} \cdot \\
\qquad\qquad \mathit{hijack}.P' \to P.B' \to B.c_B.\langle A, m\rangle \textbf{ in } tr \wedge \\
\qquad\qquad ((P' = P) \vee \mathit{Dishonest}(P)) \wedge ((B' = B) \vee \mathit{Dishonest}(B')) \vee \\
\qquad\quad \exists A' : \hat{R}_i; P' : \widehat{\mathit{Proxy}}; c_P : \mathit{Connection} \cdot \\
\qquad\qquad \mathit{hijack}.A' \to A.P' \to P.c_P.\langle m, B\rangle \textbf{ in } tr \wedge \\
\qquad\qquad ((A' = A) \vee \mathit{Dishonest}(A)) \wedge ((P' = P) \vee \mathit{Dishonest}(P'))\,.
\end{array}
$$

In Appendix B.2 we always assume that $P' = P$, since honest agents only accept messages from honest proxies, and only send messages to honest proxies, so $\mathit{Dishonest}(P)$ and $\mathit{Dishonest}(P')$ never hold.

6.2 Chaining theorem

We make the following observations of the overall channel through a multiplexing proxy.

Observation 6.7. The intruder cannot redirect messages using the channel to the proxy. Subject to the collapsing cases described earlier, the channel to the proxy satisfies *NR*.

Observation 6.8. The intruder cannot re-ascribe messages using the channel from the proxy. Subject to the collapsing cases described earlier, the channel from the proxy satisfies *NRA*.

Theorem 6.9 (Chaining theorem)**.** *If roles R_i and R_j communicate through multiplexing proxies (i.e. $\mathit{Proxies}(R_i \to R_j)$) on secure channels such that:*

$$
\begin{array}{l}
\mathit{ChannelSpec}_1(R_i \to \mathit{Proxy})\,, \\
\mathit{ChannelSpec}_2(\mathit{Proxy} \to R_j)\,,
\end{array}
$$

where $\mathit{ChannelSpec}_1$ and $\mathit{ChannelSpec}_2$ are channels in the hierarchy, then the overall channel (through the proxy) satisfies the channel specification:

$$
\mathit{ChannelSpec} =\downarrow (\nwarrow_m \mathit{ChannelSpec}_1 \sqcap \nearrow_m \mathit{ChannelSpec}_2)\,;
$$

where:

$$
\begin{array}{l}
\nwarrow_m (c, \mathit{nf}, \mathit{nra}, \mathit{nr}) = (c, \mathit{nf}, \mathit{nra}, 2)\,, \\
\nearrow_m (c, \mathit{nf}, \mathit{nra}, \mathit{nr}) = (c, \mathit{nf}, 2, \mathit{nr})\,,
\end{array}
$$

and $\sqcap$ is the greatest lower bound operator in the full lattice.

The proof of the chaining theorem is in Section 6.3.

Corollary 6.10. *If roles R_i and R_j communicate through multiplexing proxies (i.e. Proxies($R_i \to R_j$)) on secure channels such that:*

$$\begin{array}{l} \mathit{ChannelSpec}(R_i \to \mathit{Proxy})\,, \\ \mathit{ChannelSpec}(\mathit{Proxy} \to R_j)\,, \end{array}$$

where ChannelSpec is a channel in the hierarchy, then the overall channel (through the proxy) satisfies a channel specification ChannelSpec′ which is such that:

$$\mathit{ChannelSpec} \preccurlyeq \mathit{ChannelSpec}'\,.$$

In particular, ChannelSpec(Proxy($R_i \to R_j$)) holds.

A simple case analysis in the case of the multiplexing proxies shows that:

$$\mathit{ChannelSpec}' = \mathit{ChannelSpec}\,.$$

Example 6.11. The channel to the proxy satisfies $C \wedge NF \wedge NRA^- \wedge NR^-$, and the channel from the proxy satisfies $C \wedge NRA^- \wedge NR^-$.

$$\begin{array}{l} \nwarrow_m (C \wedge NF \wedge NRA^- \wedge NR^-) = C \wedge NF \wedge NRA^- \wedge NR\,, \\ \nearrow_m (C \wedge NRA \wedge NR^-) = C \wedge NRA \wedge NR^-\,. \end{array}$$

The greatest lower bound of these two points is $C \wedge NRA^- \wedge NR^-$; this is the greatest lower bound of the two channels. This is the same result as the simple proxies.

The intruder cannot re-ascribe messages to honest agents because the channel from the proxy is only re-ascribable with dishonest identities; even though the channel from the proxy is fakeable, both channels are confidential, so the intruder cannot learn the message and fake it to effect a re-ascribe. The intruder can redirect messages that are sent to him.

Example 6.12. The channel to the proxy satisfies $NF \wedge NRA^-$, and the channel from the proxy satisfies $NF \wedge NRA^- \wedge NR^-$.

$$\begin{array}{l} \nwarrow_m (NF \wedge NRA^-) = NF \wedge NRA^- \wedge NR\,, \\ \nearrow_m (NF \wedge NRA^- \wedge NR^-) = NF \wedge NRA \wedge NR^-\,. \end{array}$$

The greatest lower bound of these two points is $NF \wedge NRA^- \wedge NR^-$; this channel is stronger than the greatest lower bound of the two channels. This result is different to the simple proxy result (which is just $NF \wedge NRA^-$).

Neither channel is confidential, so the overall channel is not confidential. However, because neither channel is fakeable, the intruder cannot overhear messages and fake them to re-ascribe or redirect messages. He cannot therefore redirect messages using the channel to the proxy, and cannot re-ascribe messages using the channel from the proxy.

Example 6.13. The channel to the proxy satisfies $C \wedge NF \wedge NRA \wedge NR^{-}$, and the channel from the proxy satisfies $C \wedge NF \wedge NRA^{-} \wedge NR$.

$$\begin{aligned} \nwarrow_m (C \wedge NF \wedge NRA^{-} \wedge NR) &= C \wedge NF \wedge NRA \wedge NR\,, \\ \nearrow_m (C \wedge NF \wedge NRA \wedge NR^{-}) &= C \wedge NF \wedge NRA \wedge NR\,. \end{aligned}$$

The greatest lower bound of these two points is the top channel; this is stronger than both channels.

The intruder cannot redirect messages, nor can he re-ascribe messages on the overall channel because these activities are blocked on the only channel that allows them.

The list of resultant channels for every instance of the chaining theorem is shown in Figure 5 (in Appendix B.2).

6.3 An automated proof of the chaining theorem

As before, each instance of Theorem 6.9 is relatively simple to prove; the proof mechanism is identical to that for the simple proxies, only the details of the proof are different. As in the previous section, we first prove an example instance of the theorem, then we describe the changes to the automated proof.

Lemma 6.14. *If roles R_i and R_j communicate through multiplexing proxies (i.e. Proxies$(R_i \to R_j)$) on secure channels such that:*

$$\begin{aligned} &(NF \wedge NRA^{-})(R_i \to Proxy)\,, \\ &(NF \wedge NRA^{-} \wedge NR^{-})(Proxy \to R_j)\,, \end{aligned}$$

then $(NF \wedge NRA^{-} \wedge NR^{-})(Proxy(R_i \to R_j))$.

Proof. We use the *Proxies* property and the alternative specifications of the channels to and from the proxy to show that the alternative form of $(NF \wedge NRA^{-} \wedge NR^{-})(Proxy(R_i \to R_j))$ holds; i.e.

$$\begin{aligned} &\forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\ &\quad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\ &\qquad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\ &\qquad \exists P' : \widehat{Proxy}; B' : \hat{R}_j; c_B : Connection \cdot \\ &\qquad\quad hijack.P' \to P.B' \to B.c_B.\langle A, m\rangle \textbf{ in } tr \wedge \qquad (\dagger) \\ &\qquad\quad ((P' = P) \vee Dishonest(P)) \wedge ((B' = B) \vee Dishonest(B')) \vee \\ &\qquad \exists A' : \hat{R}_i; P' : \widehat{Proxy}; c_P : Connection \cdot \\ &\qquad\quad hijack.A' \to A.P' \to P.c_P.\langle m, B\rangle \textbf{ in } tr \wedge \\ &\qquad\quad ((A' = A) \vee Dishonest(A)) \wedge ((P' = P) \vee Dishonest(P'))\,, \end{aligned}$$

for all valid system traces tr such that for all prefixes $tr' \leqslant tr$:

$$\begin{aligned} &Proxies(R_i \to R_j)(tr')\,, \\ &(NF \wedge NRA^{-})(R_i \to Proxy)(tr')\,, \\ &(NF \wedge NRA^{-} \wedge NR^{-})(Proxy \to R_j)(tr')\,. \end{aligned}$$

Let A and B be agents playing roles R_i and R_j, P be a multiplexing proxy, c_B a connection and m an application-layer message. Suppose that the event $\mathit{receive}.B.c_B.P.\langle m, A\rangle$ occurs in tr; the network rule $\mathcal{N}_4$ implies the existence of one of several events earlier in the trace. The set of possible events is limited by $NF \wedge NRA^- \wedge NR^-$, which holds on the channel from the proxy:

1. $\exists c_P : \mathit{Connection} \cdot \mathit{send}.P.c_P.B.\langle A, m\rangle$ **in** tr ;
2. $\exists P' : P\widehat{\mathit{roxy}}; B' : \hat{R}_j \cdot \mathit{hijack}.P' \rightarrow P.B' \rightarrow B.c_B.\langle A, m\rangle$ **in** $tr \wedge$
   $((P' = P) \vee \mathit{Dishonest}(P)) \wedge ((B' = B) \vee \mathit{Dishonest}(B'))$.

The second of these disjuncts is already in the correct form for (†), so we only need to investigate the first.

Suppose that the event $\mathit{send}.P.c_P.B.\langle A, m\rangle$ precedes the receive event in the trace tr, for some connection identifier c_P. $\mathit{Proxies}(R_i \rightarrow R_j)$ implies $\mathit{Proxy}(P)$, and so $\mathit{receive}.P.c'_P.A.\langle m, B\rangle$ occurs earlier in the trace, for some connection c'_P.

We use network rule $\mathcal{N}_4$ again; this implies the existence of one of several events earlier in the trace, and, as before, these possibilities are limited by $NF \wedge NRA^-$, which holds on the channel to the proxy.

1. $\exists c_A : \mathit{Connection} \cdot \mathit{send}.A.c_A.P.\langle m, B\rangle$ **in** tr ;
2. $\exists A' : \hat{R}_i; P' : P\widehat{\mathit{roxy}} \cdot \mathit{hijack}.A' \rightarrow A.P' \rightarrow P.c_P.\langle m, B\rangle \wedge$
   $((A' = A) \vee \mathit{Dishonest}(A)) \wedge ((P' = P) \vee \mathit{Dishonest}(P'))$.

These disjuncts match those in (†).

We have shown that if an agent B receives a message from agent A via a multiplexing proxy, then the set of events that may precede this receive event is equal to those allowed by the alternative form of the proxy channel specification $NF \wedge NRA^- \wedge NR^-$ on the channel $R_i \rightarrow R_j$. Therefore $(NF \wedge NRA^- \wedge NR^-)(\mathit{Proxy}(R_i \rightarrow R_j))$ holds. □

There are, again, 121 instances of this theorem to prove. In order to prove these instances we adapt the automated proof of the simple chaining theorem; we describe these changes below, and the script is shown in Appendix C.

6.3.1 Deriving the full set of trace patterns

We derive the full set of trace patterns in the same way as before: we do not assume that either the channel to the proxy or the channel from the proxy satisfy any secure channel specifications. Suppose that tr is a valid system trace such that for all prefixes $tr' \leqslant tr$, $\mathit{Proxies}(R_i \rightarrow R_j)(tr')$ holds.

Let A and B be agents playing roles $\hat{R}_i$ and $\hat{R}_j$, P be a multiplexing proxy, c_B a connection and m an application-layer message. Suppose that $\mathit{receive}.B.c_B.P.\langle A, m\rangle$ occurs in tr; the network rule $\mathcal{N}_4$ implies the existence of one of several events earlier in the trace.

1. $\mathit{send}.P.c_P.B.\langle A, m\rangle$ **in** tr for some connection c_P;
2. $\mathit{fake}.P.B.c_B.\langle A, m\rangle$ **in** tr ;
3. $\exists P' : \widehat{Proxy}; B' : \hat{R}_j \cdot \mathit{hijack}.P' \rightarrow P.B' \rightarrow B.c_B.\langle A, m\rangle$ **in** tr .

There are three different possibilities: either the proxy sent the message to B, the intruder faked the message to B (with the proxy's identity), or the intruder hijacked a message sent by a different proxy to agent B'. Each of these events leads to different trace patterns, so we investigate them independently.

1. The event $\mathit{send}.P.c_P.B.\langle A, m\rangle$ occurs in the trace for some connection c_P. Since B accepts this message, $\mathit{Proxies}(R_i \rightarrow R_j)$ implies that $\mathit{Proxy}(P)$ holds for tr, and so P must previously have received a message of the form $\langle m, B\rangle$. The event $\mathit{receive}.P.c'_P.A.\langle m, B\rangle$ occurs earlier in the trace, for some connection c'_P.

   We apply network rule $\mathcal{N}_4$ again; this implies the existence of one of several events earlier in the trace.

   (a) $\mathit{send}.A.c_A.P.\langle m, B\rangle$ **in** tr for some connection c_A; the trace has the following pattern:

   $$tr_1 \mathrel{\widehat{=}} \langle \mathit{send}.A.c_A.P.\langle m, B\rangle,\ \mathit{receive}.P.c'_P.A.\langle m, B\rangle,\\ \mathit{send}.P.c_P.B.\langle A, m\rangle,\ \mathit{receive}.B.c_B.P.\langle A, m\rangle\rangle\ .$$

   (b) $\mathit{fake}.A.P.c'_P.\langle m, B\rangle$ **in** tr ; the trace has the following pattern:

   $$tr_2 \mathrel{\widehat{=}} \langle \mathit{fake}.A.P.c'_P.\langle m, B\rangle,\ \mathit{receive}.P.c'_P.A.\langle m, B\rangle,\\ \mathit{send}.P.c_P.B.\langle A, m\rangle,\ \mathit{receive}.B.c_B.P.\langle A, m\rangle\rangle\ .$$

   The intruder does not fake messages with his own identity, so in this trace pattern we assume that A is honest.

   (c) $\exists A' : \hat{R}_i; P' : \widehat{Proxy} \cdot \mathit{hijack}.A' \rightarrow A.P' \rightarrow P.c'_P.\langle m, B\rangle$ **in** tr . The intruder can only hijack messages that were previously sent ($\mathcal{N}_2$), so A' must have sent the message to P' earlier in the trace (in some connection $c_{A'}$). The trace has the following pattern:

   $$tr_3 \mathrel{\widehat{=}} \langle \mathit{send}.A'.c_{A'}.P'.\langle m, B\rangle,\\ \mathit{hijack}.A' \rightarrow A.P' \rightarrow P.c'_P.\langle m, B\rangle,\ \mathit{receive}.P.c'_P.A.\langle m, B\rangle,\\ \mathit{send}.P.c_P.B.\langle A, m\rangle,\ \mathit{receive}.B.c_B.P.\langle A, m\rangle\rangle\ .$$

We can safely block the intruder from hijacking his own messages, so in this trace pattern we assume that A' is honest. The intruder does not gain anything by redirecting the message to a different proxy (since all proxies are honest, and all proxies behave in the same way), so we assume that $P' = P$. If $A' = A$ then this trace pattern just shows a replay on the channel to the proxy; because none of our channels prevent replays, we are not interested in whether or not the intruder has this capability; we assume that $A' \neq A$.

2. The event $\mathit{fake}.P.B.c_B.\langle A, m\rangle$ occurs in the trace; the trace has the following pattern:

$$tr_4 \mathrel{\widehat{=}} \langle \mathit{fake}.P.B.c_B.\langle A, m\rangle,\ \mathit{receive}.B.c_B.P.\langle A, m\rangle\rangle\ .$$

   In this trace pattern, we assume that A is honest.

3. The event $\mathit{hijack}.P' \rightarrow P.B' \rightarrow B.c_B.\langle A, m\rangle$ occurs in the trace. Since all proxies are honest, and all behave in the same way, the intruder does not gain anything by changing the identity of the sender of the message; we assume that $P' = P$. We apply $\mathcal{N}_2$ again: the proxy P must have sent the message to B' earlier in the trace (in some connection c_P). So far, the trace has the pattern:

$$\begin{aligned} s \mathrel{\widehat{=}} \langle & \mathit{send}.P.c_P.B'.\langle A, m\rangle, \\ & \mathit{hijack}.P.B' \rightarrow B.c_B.\langle A, m\rangle,\ \mathit{receive}.B.c_B.P.\langle A, m\rangle\rangle\ . \end{aligned}$$

   P is a multiplexing proxy, so $\mathit{Proxies}(R_i \rightarrow R_j)$ applies again, and implies that $\mathit{Proxy}(P)$ holds for tr, and so P must previously have received a message of the form $\langle m, B'\rangle$ from A (in some connection c'_P). The trace now has the following pattern:

$$\begin{aligned} s \mathrel{\widehat{=}} \langle & \mathit{receive}.P.c'_P.A.\langle m, B'\rangle,\ \mathit{send}.P.c_P.B'.\langle A, m\rangle, \\ & \mathit{hijack}.P.B' \rightarrow B.c_B.\langle A, m\rangle,\ \mathit{receive}.B.c_B.P.\langle A, m\rangle\rangle\ . \end{aligned}$$

   We apply network rule $\mathcal{N}_4$ again; this implies the existence of one of several events earlier in the trace.

   (a) $\mathit{send}.A.c_A.P.\langle m, B'\rangle$ **in** tr for some connection c_A; the trace has the following pattern:

$$\begin{aligned} tr_5 \mathrel{\widehat{=}} \langle & \mathit{send}.A.c_A.P.\langle m, B'\rangle, \\ & \mathit{receive}.P.c'_P.A.\langle m, B'\rangle,\ \mathit{send}.P.c_P.B'.\langle A, m\rangle, \\ & \mathit{hijack}.P.B' \rightarrow B.c_B.\langle A, m\rangle,\ \mathit{receive}.B.c_B.P.\langle A, m\rangle\rangle\ . \end{aligned}$$

   As before, we assume that A is honest and that $B' \neq B$.

(b) $\mathit{fake}.A.P.c'_P.\langle m, B'\rangle$ **in** tr. If the intruder can fake the message $\langle m, B'\rangle$, he can also fake the message $\langle m, B\rangle$; this means that the hijack event on the channel from the proxy is unnecessary. We ignore this trace pattern.

(c) $\exists A' : \hat{R}_i; P' : \widehat{Proxy} \cdot \mathit{hijack}.A' \rightarrow A.P' \rightarrow P.c'_P.\langle m, B'\rangle$ **in** tr. As before, we assume that $P' = P$. We apply $\mathcal{N}_2$ again: the agent A' must have sent the message to P (in some connection $c_{A'}$) in order for the intruder to hijack it. The trace now has the following pattern:

$$\begin{aligned} tr_5 \mathrel{\widehat{=}} \langle & \mathit{send}.A'.c_{A'}.P.\langle m, B'\rangle,\ \mathit{hijack}.A' \rightarrow A.P.c'_P.\langle m, B'\rangle, \\ & \mathit{receive}.P.c'_P.A.\langle m, B'\rangle,\ \mathit{send}.P.c_P.B'.\langle A, m\rangle, \\ & \mathit{hijack}.P.B' \rightarrow B.c_B.\langle A, m\rangle,\ \mathit{receive}.B.c_B.P.\langle A, m\rangle\rangle\,. \end{aligned}$$

We assume that A' is honest and that $A' \neq A$ and $B' \neq B$.

6.3.2 Applying the channel properties

The properties on the channels to and from the proxy are applied in exactly the same way as for the simple proxies. The result of this stage of the automated proof is a set of trace patterns $\{tr_1, tr_2, \ldots, tr_n\}$ that are allowed on the resultant channel.

6.3.3 Determining the resultant channel specification

We determine the channel property that the resultant channel satisfies in the same way as we do in the automated proof of the simple chaining theorem. The resultant channel is confidential if and only if the channels to and from the proxy are confidential; we call this value c. We determine the remaining three co-ordinate points in the lattice by looking at the allowed trace patterns.

Each of these trace patterns demonstrates that the intruder can perform a particular event. We present below a summary of these patterns; each case shows the initial and final events in the pattern, and then describes which activity this demonstrates. This mapping allows us to assign a tuple (nf, nra, nr) to each trace pattern.

When the final event is $\mathit{receive}.B.c_B.P.\langle A, m\rangle$ (agent B receives a message from the proxy that appears to have been sent on the honest agent A's behalf):

$\boldsymbol{send}.A.c_A.P.\langle m, B\rangle$ this does not demonstrate any activity: $(1, 2, 2)$;

$\boldsymbol{send}.A.c_A.P.\langle m, I\rangle$ the intruder redirected a message that was intended for a dishonest agent: $(1, 2, 1)$;

send$.A.c_A.P.\langle m, B' \rangle$ the intruder redirected a message that was intended for an honest agent: $(1, 2, 0)$;

send$.A'.c_{A'}.P.\langle m, B \rangle$ the intruder re-ascribed a message to an honest agent: $(1, 0, 2)$;

send$.A'.c_{A'}.P.\langle m, I \rangle$ the intruder re-ascribed a message to an honest agent, and redirected a message that was intended for a dishonest agent: $(1, 0, 1)$;

send$.A'.c_{A'}.P.\langle m, B' \rangle$ the intruder re-ascribed a message to an honest agent, and redirected a message that was intended for an honest agent: $(1, 0, 0)$;

send$.I.c_I.P.\langle m, B \rangle$ the intruder hijacked his own message to simulate a fake: $(0, 2, 2)$;

fake$.A.P.c_P.\langle m, B \rangle$ the intruder faked a message to the proxy: $(0, 2, 2)$;

fake$.P.B.c_B.\langle A, m \rangle$ the intruder faked a message from the proxy: $(0, 2, 2)$.

When the final event is *receive*$.B.c_B.P.\langle I, m \rangle$ (agent B receives a message from the proxy that appears to have been sent on the dishonest agent I's behalf):

send$.I.c_I.P.\langle m, B \rangle$ this does not demonstrate any activity: $(1, 2, 2)$;

send$.A.c_A.P.\langle m, B \rangle$ the intruder re-ascribed a message to a dishonest agent: $(1, 1, 2)$;

send$.A.c_A.P.\langle m, I \rangle$ this does not demonstrate any activity (the intruder sent a message that was sent to him): $(1, 2, 2)$;

send$.A.c_A.P.\langle m, B' \rangle$ the intruder re-ascribed a message to a dishonest agent, and redirected a message that was intended for an honest agent; however, this is only possible on a non-confidential channel, so this simulates a learn and send: $(1, 2, 2)$.

The result of applying the mapping to each of the trace patterns is a set of tuples of the form (nf_i, nra_i, nr_i) for $i = 1 \ldots n$. We take the value of the confidential co-ordinate and calculate the resultant channel specification in the following way:

$$(c, nf, nra, nr) =\downarrow (c, \mathbf{min}_{i=1}^{n}(nf_i), \mathbf{min}_{i=1}^{n}(nra_i), \mathbf{min}_{i=1}^{n}(nr_i)) .$$

By eliminating trace patterns and calculating the resultant point in the hierarchy in this manner, we prove that the alternative specification of the resultant channel holds on all valid system traces in which the channel specifications to and from the proxy hold.

7 Related work

The discussion of proxies presented in this paper is based on the work of Dilloway and Lowe in [DL08]. Other authors have specified secure channels in different ways, and to different ends, and in some cases have demonstrated similar chaining results.

In [MS94], the authors describe a calculus for secure channel establishment. They define channels that offer confidentiality ($\rightarrow\bullet$), authentication of the message sender ($\bullet\rightarrow$), or both ($\bullet\rightarrow\bullet$). The authors show that if user B trusts a third party T, and there are channels from another agent A such that $A \bullet\!\!\rightarrow T \bullet\!\!\rightarrow B$, then the agents A and B can establish a new channel $A \bullet\!\!\rightarrow B$. The authors also show that confidential channels can be chained, provided that the message sender trusts the third party. These two results agree with our chaining theorems; though our results go further as we show that many more channels can be chained. However, we cannot reason about channels when only one agent trusts the third party, as the authors of [MS94] can.

In [Boy93], Boyd defines two different types of channel: *Confidentiality*, where only the intended user (or set of users) can read the message; and *Authentication*, where only the expected user (or set of users) can write the message. In Boyd's setup channels are established by utilising existing channels, or by propagating new channels between the two users wishing to communicate, often via a trusted third party (a proxy in our notation). Boyd shows that if a user A has an authenticated channel to a third party T, and T has an authenticated channel to a user B (and if B trusts T), then an authenticated channel from A to B can be established. This agrees with our (multiplexing) result that authenticated channels can be chained; as before though, our results go further as they show that many more channels can be chained.

Some authors have tried to solve the chaining problem by modifying the secure transport layer protocol. In [SBL06] the authors propose a variant of SSL/TLS in which three connections are established: a direct connection between client and server, and two direct connections between the client and a proxy, and between the proxy and the server. The direct connection can be used for highly confidential data, while the proxy channel can be used for data that doesn't have to remain secret. In [KCC01] the authors propose adding end-to-end encryption to chains of WTLS and TLS connections so that data sent via a proxy remains confidential. However, in both these cases, data can be passed through the proxy without the proxy being able to read it; the proxy can then no longer perform any application-layer jobs it might have (such as virus scanning).

8 Conclusions and future work

We have presented two chaining theorems for secure channels. The theorems are useful because they describe ways in which secure channels might be used, and they allow users of our secure channel specifications to calculate the properties of the overall channel through a proxy very simply. One can easily tell whether or not the chained form of two channels still provides a particular property. In particular, we have shown that the channels defined in [DL08] are invariant under chaining through a proxy, provided that the proxy is trustworthy.

In [DL08] we also present a session property; a session channel guarantees that all the messages received in a connection were sent in a single connection. We also specify a stream property which guarantees the session property, and that the messages were sent in the same order as that in which they were received. We propose to investigate whether the session and stream properties are invariant under chaining. It seems likely that this is the case (assuming that whenever the proxy receives several messages in a single session he forwards them in a single session, in the same order).

We also intend to investigate the effect of multiple chaining of secure channels. When agents playing role R_i send messages to agents playing role R_j through a multiplexing proxy, the elevation function is different on the channels to and from the proxy. If the chaining is set up as

$$R_i \to \mathit{Proxy} \to \mathit{Proxy}' \to R_j\,,$$

it is not clear what properties the channel through the two different proxies satisfies.

Using the theorems in this paper we could calculate the properties of the overall channel in two different ways: by calculating the resultant channel over the first two connections, then using this result to calculate the result of the overall chain, or by calculating the result of the last two connections first.[7]

$$\begin{array}{l}\downarrow (\nwarrow_m \mathit{Chain}(R_i \to \mathit{Proxy} \to \mathit{Proxy}') \sqcap \nearrow_m (\mathit{Proxy}' \to R_j))\,, \text{ or}\\ \downarrow (\nwarrow_m (R_i \to \mathit{Proxy}) \sqcap \nearrow_m \mathit{Chain}(\mathit{Proxy} \to \mathit{Proxy}' \to R_j))\,,\end{array}$$

where:

$$\begin{array}{l}\mathit{Chain}(R_i \to \mathit{Proxy} \to \mathit{Proxy}') =\\ \quad \downarrow (\nwarrow_m (R_i \to \mathit{Proxy}) \sqcap \nearrow_m (\mathit{Proxy} \to \mathit{Proxy}'))\,, \text{ and}\\ \mathit{Chain}(\mathit{Proxy} \to \mathit{Proxy}' \to R_j) =\\ \quad \downarrow (\nwarrow_m (\mathit{Proxy} \to \mathit{Proxy}') \sqcap \nearrow_m (\mathit{Proxy}' \to R_j))\,.\end{array}$$

[7] In these examples, when we write a channel such as $R_i \to R_j$, we are of course referring to the channel properties satisfied by this channel, rather than the channel itself.

Because the elevation functions ($\nwarrow_m$ and $\nearrow_m$) are not the same, in most cases these calculations will give different results. For this reason we believe that the overall channel is likely to satisfy the following specification:

$$\downarrow (\nwarrow_m (R_i \to \mathit{Proxy}) \sqcap (\mathit{Proxy} \to \mathit{Proxy}') \sqcap \nearrow_m (\mathit{Proxy}' \to R_j)) .$$

The specifications of the channels to the first proxy and from the last proxy are elevated in the usual way, but there is no elevation on the intermediate channel. It is easy to see how to generalise this result to longer chains.

Acknowledgements

I would like to thank Gavin Lowe for many useful discussions. This work is funded by the US Office of Naval Research.

References

[BL03] P. Broadfoot and G. Lowe. On distributed security transactions that use secure transport protocols. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pages 141–151, 2003.

[Boy93] C. Boyd. Security architectures using formal methods. *IEEE Journal on Selected Areas in Communications*, 11(5):694–701, 1993.

[DL08] C. Dilloway and G. Lowe. Specifying secure channels. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium*, 2008.

[DY83] D. Dolev and A.C. Yao. On security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[KCC01] E. Kwon, Y. Cho, and K. Chae. Integrated transport layer security: End-to-end security model between WTLS and TLS. *Proceedings of the The 15th International Conference on Information Networking*, 2001.

[MS94] Ueli Maurer and Pierre Schmid. A calculus for secure channel establishment in open networks. In *Computer Security — ESORICS 94*, volume 875 of *Lecture Notes in Computer Science*, pages 175–192. Springer-Verlag, November 1994.

[OAS05] OASIS Security Services Technical Committee. *Assertions and Protocols for the Security Assertion Markup Language (SAML) V2.0*, 2005. Available from http://www.oasis-open.org/committees/security/.

[RSG+01] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and A.W. Roscoe. *The Modelling and Analysis of Security Protocols.* Addison-Wesley, 2001.

[SBL06] Y. Song, K. Beznosov, and V. Leung. Multiple-channel security architecture and its implementation over SSL. *EURASIP Journal on Wireless Communications and Networking*, 2006(2):78–78, 2006.

[Vis06] Visa International Service Association. *Verified by Visa System Overview External Version 1.0.2*, 2006. Available from https://partnernetwork.visa.com/vpn/global/category.do.

[WSF+03] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for grid services. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 48–57, 2003.

A Alternative specifications for proxy channels

A.1 Simple proxies

Definition A.1 ($Alt(\bot)$)**.**

$$
\begin{array}{l}
Alt(\bot)(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : Connection \cdot fake.A.P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\quad fake.P_{(A,B)}.B.c_B.m \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad hijack.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad hijack.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m \textbf{ in } tr\,.
\end{array}
$$

Definition A.2 ($Alt(NF \wedge NRA^-)$)**.**

$$
\begin{array}{l}
Alt(NF \wedge NRA^-)(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m \textbf{ in } tr \vee\ .
\end{array}
$$

Definition A.3 ($Alt(NF \wedge NRA^- \wedge NR^-)$)**.**

$$
\begin{array}{l}
Alt(NF \wedge NRA^- \wedge NR^-)(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m \textbf{ in } tr \vee\ .
\end{array}
$$

Definition A.4 ($Alt(NF \wedge NRA^- \wedge NR)$)**.**

$$
\begin{array}{l}
Alt(NF \wedge NRA^- \wedge NR)(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; P_{(A',B)} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P_{(A',B)} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A',B)} \rightarrow P_{(A,B)}.B.c_B.m \textbf{ in } tr\,.
\end{array}
$$

Definition A.5 ($Alt(C \wedge NR^-)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NR^-)(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy_{(R_i,R_j)}) \wedge C(Proxy_{(R_i,R_j)} \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : Connection \cdot fake.A.P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\quad fake.P_{(A,B)}.B.c_B.m \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m \textbf{ in } tr\,.
\end{array}
$$

Definition A.6 ($Alt(C \wedge NRA^- \wedge NR^-)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NRA^- \wedge NR^-)(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy_{(R_i,R_j)}) \wedge C(Proxy_{(R_i,R_j)} \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : Connection \cdot fake.A.P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\quad fake.P_{(A,B)}.B.c_B.m \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m \textbf{ in } tr \vee\ .
\end{array}
$$

Definition A.7 ($Alt(C \wedge NRA \wedge NR^{-})$)**.**

$$
\begin{array}{l}
Alt(C \wedge NRA \wedge NR^{-})(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy_{(R_i,R_j)}) \wedge C(Proxy_{(R_i,R_j)} \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : Connection \cdot fake.A.P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\quad fake.P_{(A,B)}.B.c_B.m \textbf{ in } tr \vee \\
\qquad\quad \exists B' : \hat{R}_j; P_{(A,B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A.P_{(A,B')} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A,B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m \textbf{ in } tr \vee \ .
\end{array}
$$

Definition A.8 ($Alt(C \wedge NF \wedge NRA^{-} \wedge NR^{-})$)**.**

$$
\begin{array}{l}
Alt(C \wedge NF \wedge NRA^{-} \wedge NR^{-})(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy_{(R_i,R_j)}) \wedge C(Proxy_{(R_i,R_j)} \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; P_{(A',B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P_{(A',B')} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A',B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m \textbf{ in } tr \vee \ .
\end{array}
$$

Definition A.9 ($Alt(C \wedge NF \wedge NRA^{-} \wedge NR)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NF \wedge NRA^{-} \wedge NR)(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy_{(R_i,R_j)}) \wedge C(Proxy_{(R_i,R_j)} \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i P_{(A',B)} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P_{(A',B)} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A',B)} \rightarrow P_{(A,B)}.B.c_B.m \textbf{ in } tr \, .
\end{array}
$$

Definition A.10 ($Alt(C \wedge NF \wedge NRA \wedge NR^{-})$)**.**

$$
\begin{array}{l}
Alt(C \wedge NF \wedge NRA \wedge NR^{-})(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy_{(R_i,R_j)}) \wedge C(Proxy_{(R_i,R_j)} \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr \vee \\
\qquad\quad \exists B' : \hat{R}_j; P_{(A,B')} : \widehat{Proxy}_{(R_i,R_j)}; c_P : Connection \cdot \\
\qquad\qquad ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A.P_{(A,B')} \rightarrow P_{(A,B)}.c_P.m \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P_{(A,B')} \rightarrow P_{(A,B)}.B' \rightarrow B.c_B.m \textbf{ in } tr\,.
\end{array}
$$

Definition A.11 ($Alt(C \wedge NF \wedge NRA \wedge NR)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NF \wedge NRA \wedge NR)(Proxy_{(R_i,R_j)})(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy_{(R_i,R_j)}) \wedge C(Proxy_{(R_i,R_j)} \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P_{(A,B)} : \widehat{Proxy}_{(R_i,R_j)}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P_{(A,B)}.m \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P_{(A,B)}.m \textbf{ in } tr\,.
\end{array}
$$

A.2 Multiplexing proxies

Definition A.12 ($Alt(\perp)$)**.**

$$
\begin{array}{l}
Alt(\perp)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : Connection \cdot fake.A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad fake.P.B.c_B.\langle A, m\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; c_P : Connection \cdot \\
\qquad\qquad hijack.A' \rightarrow A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\qquad hijack.P.B' \rightarrow B.c_B.\langle A, m\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.13 ($Alt(NF \wedge NRA^{-})$)**.**

$$
\begin{array}{l}
Alt(NF \wedge NRA^{-})(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P.B' \rightarrow B.c_B.\langle A, m\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.14 ($Alt(NF \wedge NRA^- \wedge NR^-)$)**.**

$$
\begin{array}{l}
Alt(NF \wedge NRA^- \wedge NR^-)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P.B' \rightarrow B.c_B.\langle A, m\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.15 ($Alt(NF \wedge NRA^- \wedge NR)$)**.**

$$
\begin{array}{l}
Alt(NF \wedge NRA^- \wedge NR)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P.c_P.\langle m, B\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.16 ($Alt(C \wedge NR^-)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NR^-)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy) \wedge C(Proxy \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : Connection \cdot fake.A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad fake.P.B.c_B.\langle A, m\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; c_P : Connection \cdot \\
\qquad\qquad ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P.B' \rightarrow B.c_B.\langle A, m\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.17 ($Alt(C \wedge NRA^- \wedge NR^-)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NRA^- \wedge NR^-)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy) \wedge C(Proxy \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : Connection \cdot fake.A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad fake.P.B.c_B.\langle A, m\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P.B' \rightarrow B.c_B.\langle A, m\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.18 ($Alt(C \wedge NRA \wedge NR^-)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NRA \wedge NR^-)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy) \wedge C(Proxy \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists c_P : Connection \cdot fake.A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad fake.P.B.c_B.\langle A, m\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists B' : \hat{R}_j; c_P : Connection \cdot \\
\qquad\qquad ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.P.B' \rightarrow B.c_B.\langle A, m\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.19 ($Alt(C \wedge NF \wedge NRA^- \wedge NR^-)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NF \wedge NRA^- \wedge NR^-)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy) \wedge C(Proxy \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; B' : \hat{R}_j; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P.c_P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\qquad\quad hijack.P.B' \rightarrow B.c_B.\langle A, m\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.20 ($Alt(C \wedge NF \wedge NRA^- \wedge NR)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NF \wedge NRA^- \wedge NR)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy) \wedge C(Proxy \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists A' : \hat{R}_i; c_P : Connection \cdot \\
\qquad\qquad ((A' = A) \vee Dishonest(A)) \wedge \\
\qquad\qquad\quad hijack.A' \rightarrow A.P.c_P.\langle m, B\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.21 ($Alt(C \wedge NF \wedge NRA \wedge NR^-)$)**.**

$$
\begin{array}{l}
Alt(C \wedge NF \wedge NRA \wedge NR^-)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
\quad C(R_i \rightarrow Proxy) \wedge C(Proxy \rightarrow R_j) \wedge \\
\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr \vee \\
\qquad\quad \exists B' : \hat{R}_j; c_P : Connection \cdot \\
\qquad\qquad ((B' = B) \vee Dishonest(B')) \wedge \\
\qquad\qquad\quad hijack.P.B' \rightarrow B.c_B.\langle A, m\rangle \textbf{ in } tr\,.
\end{array}
$$

Definition A.22 ($Alt(C \wedge NF \wedge NRA \wedge NR)$)**.**

$$
\begin{aligned}
&Alt(C \wedge NF \wedge NRA \wedge NR)(Proxy(R_i, R_j))(tr) \mathrel{\widehat{=}} \\
&\quad C(R_i \to Proxy) \wedge C(Proxy \to R_j) \wedge \\
&\quad \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i; P : \widehat{Proxy}; m : Message_{App} \cdot \\
&\qquad receive.B.c_B.P.\langle A, m\rangle \textbf{ in } tr \Rightarrow \\
&\qquad\quad \exists c_A : Connection \cdot send.A.c_A.P.\langle m, B\rangle \textbf{ in } tr\,.
\end{aligned}
$$

B Chaining theorem result tables

B.1 Simple proxies

See Figure 4.

B.2 Multiplexing proxies

See Figure 5.

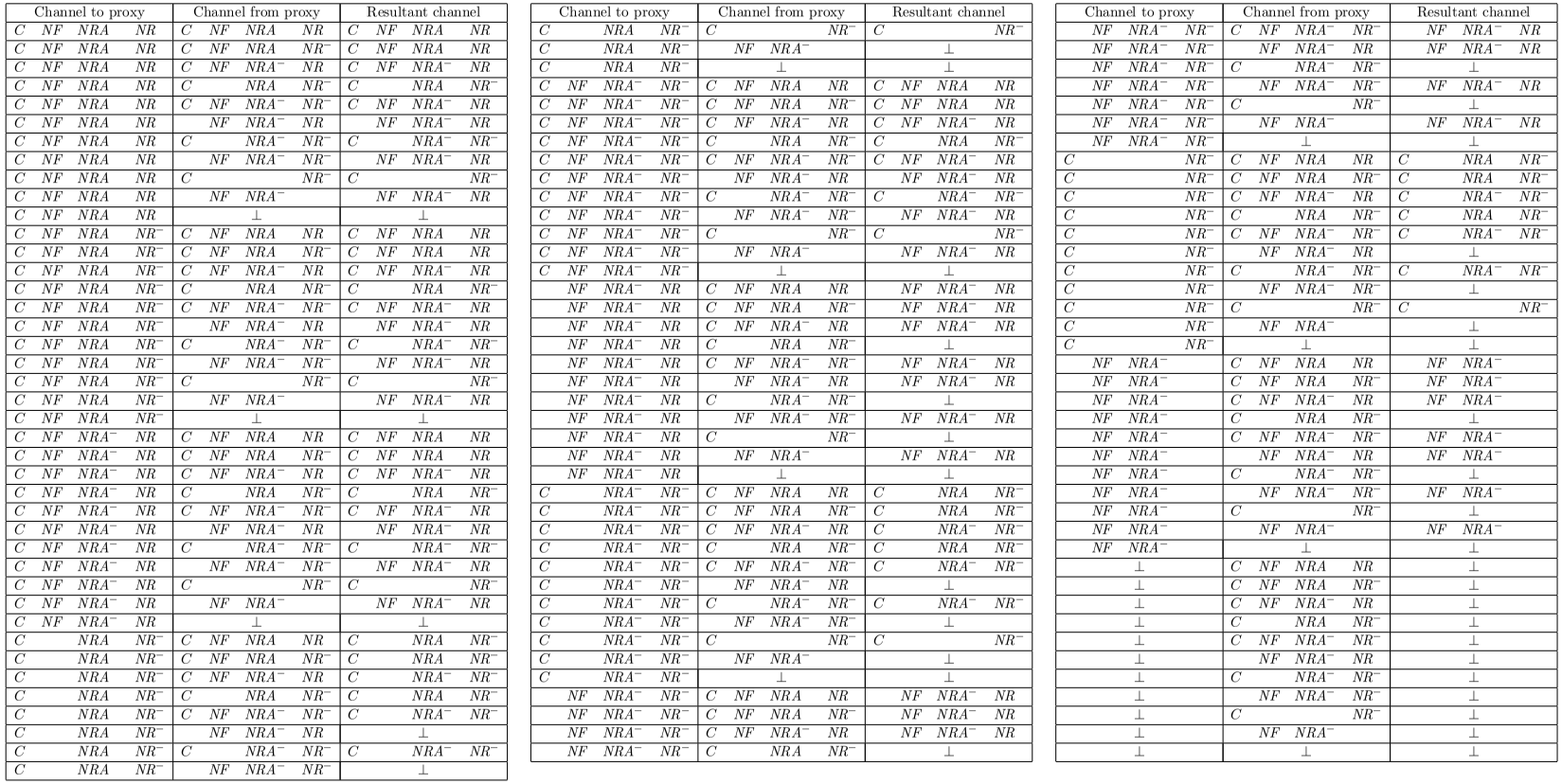


Figure 4: The table of resultant channels through a simple proxy.

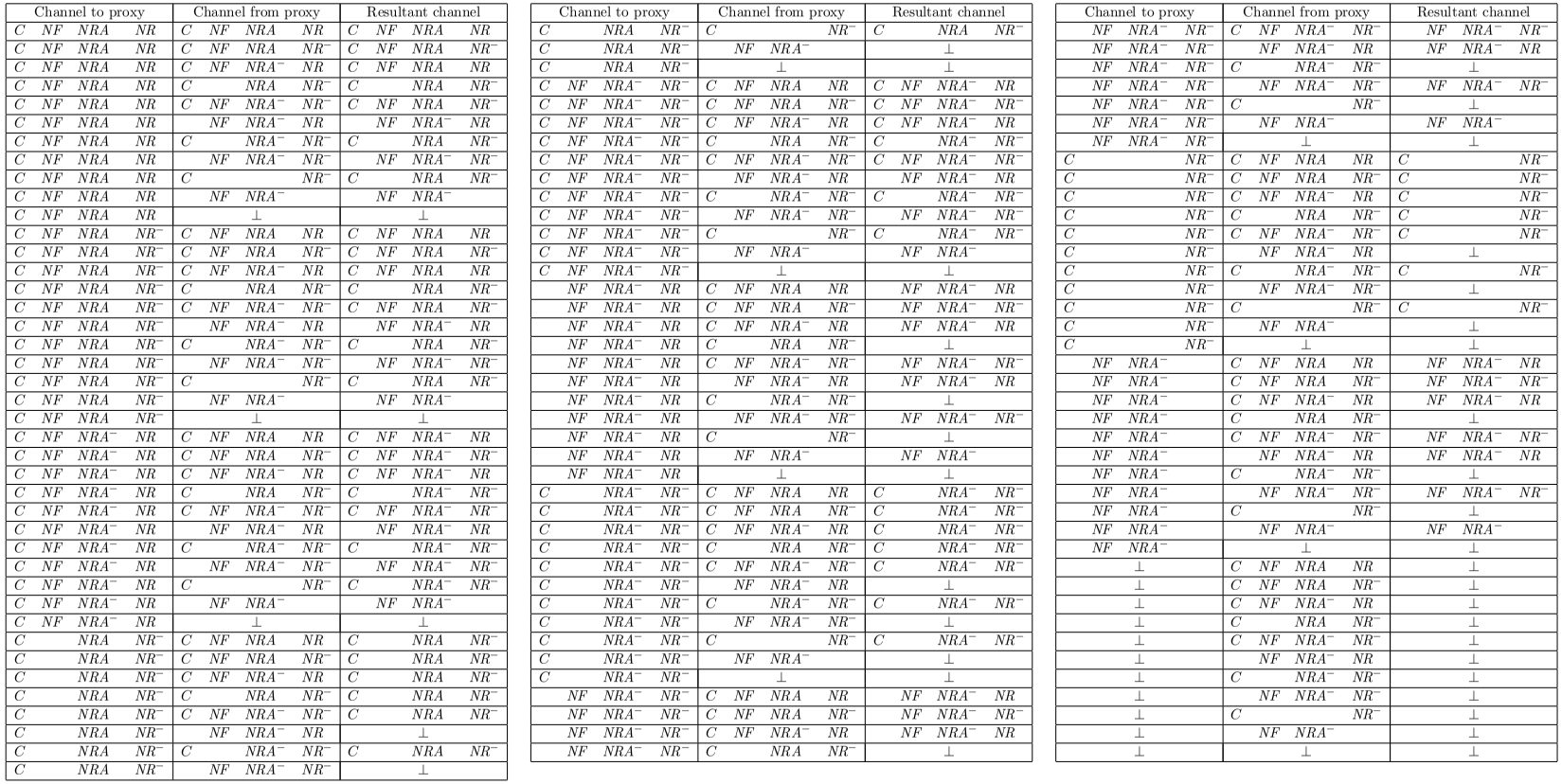


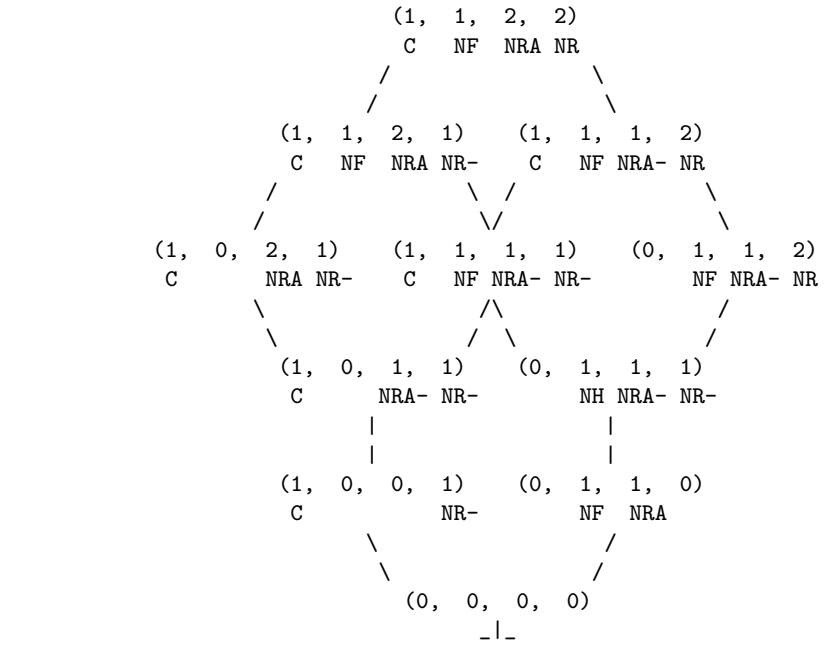
Figure 5: The table of resultant channels through a multiplexing proxy.

C Chaining theorem Haskell scripts

C.1 Subsidiary (shared) channel functions

channels.lhs

```
>module Channels (
>  Channel, ChannelComp (Less, Greater, Equal, Incomparable),
>  channelCompare, hierarchy, collapse, channelShow, glb)
>where

A channel is represented by its co-ordinates in the full lattice:
(C, NF, NRA, NR)

>type Channel = (Integer, Integer, Integer, Integer)

>data ChannelComp = Less | Equal | Greater | Incomparable
>                  deriving (Eq, Show)

Channels are compared pointwise:

>channelCompare :: Channel -> Channel -> ChannelComp
>channelCompare (c1, nf1, nra1, nr1) (c2, nf2, nra2, nr2)
> | c1 == c2 && nf1 == nf2 && nra1 == nra2 && nr1 == nr2 = Equal
> | c1 <= c2 && nf1 <= nf2 && nra1 <= nra2 && nr1 <= nr2 = Less
> | c1 >= c2 && nf1 >= nf2 && nra1 >= nra2 && nr1 >= nr2 = Greater
> | otherwise = Incomparable

The hierarchy is represented as follows:
```




```
>hierarchy :: [Channel]
```

```
>hierarchy = [(1, 1, 2, 2),
>(1, 1, 2, 1), (1, 1, 1, 2),
>(1, 0, 2, 1), (1, 1, 1, 1), (0, 1, 1, 2),
>(1, 0, 1, 1), (0, 1, 1, 1),
>(1, 0, 0, 1), (0, 1, 1, 0),
>(0, 0, 0, 0)]

Collapse a point in the lattice by applying the collapsing rules until we
reach a fixpoint

>collapse :: Channel -> Channel
>collapse c = if (c == c') then c else collapse c'
>  where c' = collapse' c

>collapse' :: Channel  -> Channel
>collapse' (0,0,x,y) = (0,0,0,0)
>collapse' (x,1,0,y) = (x,0,0,y)
>collapse' (0,1,2,x) = (0,1,1,x)
>collapse' (1,x,y,0) = (0,x,y,0)
>collapse' (1,0,x,2) = (1,0,x,1)
>collapse' c = c

Show a channel as a string

>channelShow :: Channel -> String
>channelShow (0,0,0,0) = "_|_"
>channelShow (c, nf, nra, nr) = cStr ++ nfStr ++ nraStr ++ nrStr
>  where cStr    = if c > 0 then "C" ++
>                  (if nf + nra + nr > 0 then " ^ " else "") else ""
>        nfStr   = if nf > 0 then "NF" ++
>  (if nra + nr > 0 then " ^ " else "") else ""
>        nraStr  = if nra > 0 then dashNRA ++
>  (if nr > 0 then " ^ " else "") else ""
>        nrStr   = if nr > 0 then dashNR else ""
>        dashNRA = if nra == 1 then "NRA-" else "NRA"
>        dashNR  = if nr == 1 then "NR-" else "NR"

The greatest lower bound (in the hierarchy) of two channels

>glb :: Channel -> Channel -> Channel
>glb (c1, nf1, nra1, nr1) (c2, nf2, nra2, nr2) =
>  (min c1 c2, min nf1 nf2, min nra1 nra2, min nr1 nr2)
```

C.2 Simple proxy proof script

```
>import Channels

We calculate the channel combinations where the expected result (calculated
using the elevation rules) and the actual result (calculated using the trace
patterns) differ.

>difference :: IO()
>difference = (putStr . concat) [
>  "Actual: " ++ (actual c1 c2) ++
```

```
>  "Expected: " ++ (expected c1 c2) ++ "\n" |
>   c1 <- hierarchy, c2 <- hierarchy,
>   (actual c1 c2) /= (expected c1 c2)]
```

The expected result (calculated using the elevation rules).

```
>expected :: Channel -> Channel -> String
>expected c1 c2 = "A --[" ++
>  (channelShow c1) ++ "]--> Proxy --[" ++
>  (channelShow c2) ++ "]--> B = " ++
>  (channelShow resultant) ++ "\n"
>  where resultant = collapse (glb (raiseToProxy c1) (raiseFromProxy c2))

>raiseToProxy :: Channel -> Channel
>raiseToProxy (c, nf, nra, nr) = (c, nf, nra', nr')
>  where nra' = if (nr > 0) then 2 else nra
>        nr'  = if (nr > 0) then 2 else 0

>raiseFromProxy :: Channel -> Channel
>raiseFromProxy (c, nf, nra, nr) = (c, nf, nra, nr')
>  where nr' = if (nra > 0) then 2 else nr
```

The actual result (calculated using trace patterns).

```
>actual :: Channel -> Channel -> String
>actual c1 c2 = "A --[" ++
>  (channelShow c1) ++ "]--> Proxy --[" ++
>  (channelShow c2) ++ "]--> B = " ++
>  (channelShow (match (resultant A) (resultant I) c1 c2)) ++ "\n"
>  where resultant a = (map head . toSender a . toProxy) (finalEvent a)
>        toSender a = applyChannel c1 . concat . map
>                     (initial (Sender a) (Receiver B))
>        toProxy = map preProxy . applyChannel c2 . pre
>        finalEvent a = (Receive (Proxy (Sender a, Receiver B), Receiver B))
```

We use representative identities: A, A', B, B' are honest; I is dishonest.

```
>data Identity = A | A' | B | B' | I
>  deriving (Eq, Show)
```

An agent is either a proxy between two agents, a sender, or a receiver.

```
>data Agent = Proxy (Agent, Agent) | Sender Identity | Receiver Identity
>  deriving (Eq, Show)
```

Proxies are honest if the agent they send on behalf of is honest.

```
>honest :: Agent -> Bool
>honest (Proxy (a,b)) = honest a
>honest (Sender a) = a /= I
>honest (Receiver a) = a /= I
```

It's convenient to list the senders, receivers and proxies.

```
>senders :: [Agent]
>senders = [Sender A, Sender A', Sender I]
>receivers :: [Agent]
>receivers = [Receiver B, Receiver B', Receiver I]
>proxies :: [Agent]
>proxies = [Proxy (a, b) | a <- senders, b <- receivers]
```

And to pick out the sender or receiver from a proxy.

```
>sender :: Agent -> Agent
>sender (Proxy (a, b)) = a
>receiver :: Agent -> Agent
>receiver (Proxy (a, b)) = b
```

Each event is either a send, receive, fake or hijack. We list the sender's identity first, the recipient's second.

```
>data Event = Send (Agent, Agent) |
>             Receive (Agent, Agent) |
>             Fake (Agent, Agent) |
>             Hijack (Agent, Agent, Agent, Agent)
>  deriving (Eq,Show)
```

The function pre calculates which events could have occurred immediately before the final receive event. However, we don't let the intruder fake with his own (e.g. a dishonest) identity.

```
>pre :: Event -> [[Event]]
>pre (Receive (p,b)) = sends:fakes:hijacks
>  where sends = [Send (p,b), Receive (p,b)]
>        fakes = if (honest (p)) then [Fake (p,b), Receive (p,b)]
>                       else []
>        hijacks = [[Send (p', receiver (p')),
>                            Hijack (p', p, receiver (p'), b),
>                            Receive (p,b)] | p' <- proxies]
```

If the proxy p sent a message to b, then who sent the message to p?

```
>preProxy :: [Event] -> [Event]
>preProxy (Send (p,b):xs) = Receive (sender p, p):Send (p,b):xs
>preProxy xs = xs
```

Once we know who the proxy received the message from we can work out all possible traces that would result in our original event. However:
We don't let the intruder send messages to the wrong recipient;
We don't let the intruder fake with his own identity, with the wrong sender's identity, or to the wrong recipient.

```
>initial :: Agent -> Agent -> [Event] -> [[Event]]
>initial a1 b1 (Receive (a,p):xs) = sends:fakes:hijacks
>  where sends = if (honest (a) || receiver (p) == b1) then
>                Send (a,p):Receive (a,p):xs else []
>        fakes = if (honest (a) && a == a1 && receiver(p) == b1) then
>                Fake (a,p):Receive (a,p):xs else []
```

```
>        hijacks = [[Send ((sender (p')), p'),
>                    Hijack (sender (p'),a,p',p),
>                    Receive (a,p)] ++ xs | p' <- proxies]
>initial _ _ xs = [xs]
```

We can apply a channel to the events each side of the proxy (apply only looks at the first two events in each trace).

```
>applyChannel :: Channel -> [[Event]] -> [[Event]]
>applyChannel _ [] = []
>applyChannel cs ([]:xss) = applyChannel cs xss
>applyChannel (c,nf,nra,nr) ((x:y:xs):xss) =
>  if (nfs nf (x,y) && nras nra (x,y) && nrs nr (x,y)) then
>  (x:y:xs):(applyChannel (c,nf,nra,nr) xss) else
>  (applyChannel (c,nf,nra,nr) xss)
>  where nfs 0 _ = True
>        nfs 1 ((Fake (a,b)),y) = False
>        nfs 1 _ = True
>        nras 0 _ = True
>        nras 1 (x,(Hijack (a,a',b,b'))) = (a == a') || (not $ honest(a'))
>        nras 1 _ = True
>        nras 2 (x,(Hijack (a,a',b,b'))) = (a == a')
>        nras 2 _ = True
>        nrs 0 _ = True
>        nrs 1 (x,(Hijack (a,a',b,b'))) = (b == b') || (not $ honest(b))
>        nrs 1 _ = True
>        nrs 2 (x,(Hijack (a,a',b,b'))) = (b == b')
>        nrs 2 _ = True
```

The function match takes a list of initial honest and initial dishonest events and discovers which channel they correspond to.

```
>match :: [Event] -> [Event] -> Channel -> Channel -> Channel
>match hs ds (c1,nf1,nra1,nr1) (c2,nf2,nra2,nr2) = collapse (c,nf,nra,nr)
>  where c   = if (c1 == 1 && c2 == 1) then 1 else 0
>        nf  = if (nf1 == 1 && nf2 == 1) then 1 else 0
>        nra = minimum (map trd (map hEvents hs ++ map dEvents ds))
>        nr  = minimum (map fth (map hEvents hs ++ map dEvents ds))
```

The functions hEvents and dEvents tell which channel properties a certain event implies.

```
>hEvents :: Event -> Channel
>hEvents (Send (Sender A,Proxy (Sender A,Receiver B)))    = (1,1,2,2)
>hEvents (Send (Sender A,Proxy (Sender A,Receiver I)))    = (1,1,2,1)
>hEvents (Send (Sender A',Proxy (Sender A',Receiver B)))  = (1,1,0,2)
>hEvents (Send (Sender A',Proxy (Sender A',Receiver I)))  = (1,1,0,1)
>hEvents (Send (Sender I,Proxy (Sender I,Receiver B)))    = (1,0,2,2)
>hEvents (Send (Sender I,Proxy (Sender I,Receiver I)))    = (1,0,2,2)
>hEvents (Send (Sender A,Proxy (Sender A,Receiver B')))   = (1,1,2,0)
>hEvents (Send (Sender A',Proxy (Sender A',Receiver B'))) = (1,1,0,0)
>hEvents (Send (Sender I,Proxy (Sender I,Receiver B')))   = (1,0,2,2)
>hEvents (Fake (Sender A,Proxy (Sender A,Receiver B)))    = (1,0,2,2)
>hEvents (Fake (Proxy (Sender A,Receiver B),Receiver B))  = (1,0,2,2)
```

```
>dEvents :: Event -> Channel
>dEvents (Send (Sender I,Proxy (Sender I,Receiver B)))    = (1,1,2,2)
>dEvents (Send (Sender A,Proxy (Sender A,Receiver B)))    = (1,1,1,2)
>dEvents (Send (Sender A',Proxy (Sender A',Receiver B)))  = (1,1,1,2)
>dEvents (Send (Sender A,Proxy (Sender A,Receiver I)))    = (1,1,2,2)
>dEvents (Send (Sender A',Proxy (Sender A',Receiver I)))  = (1,1,2,2)
>dEvents (Send (Sender I,Proxy (Sender I,Receiver I)))    = (1,1,2,2)
>dEvents (Send (Sender A,Proxy (Sender A,Receiver B')))   = (0,1,2,2)
>dEvents (Send (Sender A',Proxy (Sender A',Receiver B'))) = (0,1,2,2)
>dEvents (Send (Sender I,Proxy (Sender I,Receiver B')))   = (1,1,2,2)

>trd (_,_,x,_) = x
>fth (_,_,_,x) = x
```

C.3 Multiplexing proxy proof script

```
>import Channels

We calculate the channel combinations where the expected result (calculated
using the elevation rules) and the actual result (calculated using the trace
patterns) differ.

>difference :: IO()
>difference = (putStr . concat) [
>  "Actual: " ++ (actual c1 c2) ++
>  "Expected: " ++ (expected c1 c2) ++ "\n" |
>   c1 <- hierarchy, c2 <- hierarchy,
>   (actual c1 c2) /= (expected c1 c2)]

The expected result (calculated using the elevation rules).

>expected :: Channel -> Channel -> String
>expected c1 c2 = "A --[" ++
>  (channelShow c1) ++ "]--> Proxy --[" ++
>  (channelShow c2) ++ "]--> B = " ++
>  (channelShow resultant) ++ "\n"
>  where resultant = collapse (glb (raiseToProxy c1) (raiseFromProxy c2))

>raiseToProxy :: Channel -> Channel
>raiseToProxy (c, nf, nra, nr) = (c, nf, nra, 2)

>raiseFromProxy :: Channel -> Channel
>raiseFromProxy (c, nf, nra, nr) = (c, nf, 2, nr)

The actual result (calculated using trace patterns).

>actual :: Channel -> Channel -> String
>actual c1 c2 = "A --[" ++
>  (channelShow c1) ++ "]--> Proxy --[" ++
>  (channelShow c2) ++ "]--> B = " ++
>  (channelShow (match (resultant A) (resultant I) c1 c2)) ++ "\n"
>  where resultant a = (map head . toSender a . toProxy) (finalEvent a)
>        toSender a = applyChannel c1 . concat . map
```

```
>                     (initial (Sender a) (Receiver B))
>        toProxy = map preProxy . applyChannel c2 . pre
>        finalEvent a = (Receive (Proxy, Receiver B, Sender a))
```

We use representative identities: A, A', B, B' are honest; I is dishonest.

```
>data Identity = A | A' | B | B' | I
>  deriving (Eq, Show)
```

An agent is either a proxy, a sender, or a receiver.

```
>data Agent = Proxy | Sender Identity | Receiver Identity
>  deriving (Eq, Show)
```

All proxies are honest (so we never question it).

```
>honest :: Agent -> Bool
>honest (Sender a) = a /= I
>honest (Receiver a) = a /= I
```

It's convenient to list the senders and receivers.

```
>senders :: [Agent]
>senders = [Sender A, Sender A', Sender I]

>receivers :: [Agent]
>receivers = [Receiver B, Receiver B', Receiver I]
```

Each event is either a send, receive, fake or hijack. We list the sender's idenitity first, the receiver's second and the third party's (the original sender or the final recipient) third.

```
>data Event = Send (Agent, Agent, Agent) |
>             Receive (Agent, Agent, Agent) |
>             Fake (Agent, Agent, Agent) |
>             Hijack (Agent, Agent, Agent, Agent)
>  deriving (Eq,Show)
```

The function pre calculates which events could have occurred immediately before the final receive event. However, we don't let the intruder fake with his own (e.g. a dishonest) identity.

```
>pre :: Event -> [[Event]]
>pre (Receive (Proxy,b,a)) = sends:fakes:hijacks
>  where sends = [Send (Proxy,b,a), Receive (Proxy,b,a)]
>        fakes = if (honest (a)) then
>                [Fake (Proxy,b,a), Receive (Proxy,b,a)] else []
>        hijacks = [[Send (Proxy,b',a),
>                    Hijack (Proxy,b',b,a),
>                    Receive (Proxy,b,a)] | b' <- receivers]
```

If the proxy p sent a message to b, then who sent the message to p?

```
>preProxy :: [Event] -> [Event]
```

```
>preProxy (Send (Proxy,b,a):xs) = Receive (a,Proxy,b):Send (Proxy,b,a):xs
>preProxy xs = xs
```

Once we know who the proxy received the message from we can work out all possible traces that would result in our original event. However:
We don't let the intruder send messages to the wrong recipient;
We don't let the intruder fake with his own identity, with the wrong sender's identity, or to the wrong recipient.

```
>initial :: Agent -> Agent -> [Event] -> [[Event]]
>initial a1 b1 (Receive (a,Proxy,b):xs) = sends:fakes:hijacks
>  where sends = if (honest(a) || b == b1) then
>                (Send (a,Proxy,b)):(Receive (a,Proxy,b)):xs else []
>        fakes = if (honest(a) && a == a1 && b == b1) then
>                (Fake (a,Proxy,b)):(Receive (a,Proxy,b)):xs else []
>        hijacks = [[Send (a',Proxy,b),
>                    Hijack (a',a,Proxy,b),
>                    Receive (a,Proxy,b)] ++ xs | a' <- senders]
>initial _ _ xs = [xs]
```

We can apply a channel to the events each side of the proxy (apply only looks at the first two events in each trace).

```
>applyChannel :: Channel -> [[Event]] -> [[Event]]
>applyChannel _ [] = []
>applyChannel cs ([]:xss) = applyChannel cs xss
>applyChannel (c,nf,nra,nr) ((x:y:xs):xss) =
>  if (nfs nf (x,y) && nras nra (x,y) && nrs nr (x,y)) then
>  (x:y:xs):(applyChannel (c,nf,nra,nr) xss) else
>  (applyChannel (c,nf,nra,nr) xss)
>  where nfs 0 _ = True
>        nfs 1 ((Fake (a,Proxy,b)),y) = False
>        nfs 1 ((Fake (Proxy,b,a)),y) = False
>        nfs 1 _ = True
>        nras 0 _ = True
>        nras 1 (x,(Hijack (a,a',Proxy,b))) = (a == a') || (not $ honest(a'))
>        nras 1 (x,(Hijack (Proxy,b,b',a))) = True
>        nras 1 _ = True
>        nras 2 (x,(Hijack (a,a',Proxy,b))) = (a == a')
>        nras 2 (x,(Hijack (Proxy,b,b',a))) = True
>        nras 2 _ = True
>        nrs 0 _ = True
>        nrs 1 (x,(Hijack (a,a',Proxy,b))) = True
>        nrs 1 (x,(Hijack (Proxy,b,b',a))) = (b == b') || (not $ honest(b))
>        nrs 1 _ = True
>        nrs 2 (x,(Hijack (a,a',Proxy,b))) = True
>        nrs 2 (x,(Hijack (Proxy,b,b',a))) = (b == b')
>        nrs 2 _ = True
```

The function match takes a list of initial honest and initial dishonest events and discovers which channel they correspond to.

```
>match :: [Event] -> [Event] -> Channel -> Channel -> Channel
>match hs ds (c1,nf1,nra1,nr1) (c2,nf2,nra2,nr2) = collapse (c,nf,nra,nr)
```

```
>  where c   = if (c1 == 1 && c2 == 1) then 1 else 0
>        nf  = if (nf1 == 1 && nf2 == 1) then 1 else 0
>        nra = minimum (map trd (map hEvents hs ++ map dEvents ds))
>        nr  = minimum (map fth (map hEvents hs ++ map dEvents ds))
```

The functions hEvents and dEvents tell which channel properties a certain event implies.

```
>hEvents :: Event -> Channel
>hEvents (Send (Sender A,Proxy,Receiver B))   = (1,1,2,2)
>hEvents (Send (Sender A,Proxy,Receiver I))   = (1,1,2,1)
>hEvents (Send (Sender A,Proxy,Receiver B'))  = (1,1,2,0)
>hEvents (Fake (Sender A,Proxy,Receiver B))   = (1,0,2,2)
>hEvents (Fake (Proxy,Receiver B,Sender A))   = (1,0,2,2)
>hEvents (Send (Sender A',Proxy,Receiver B))  = (1,1,0,2)
>hEvents (Send (Sender I,Proxy,Receiver B))   = (1,0,2,2)
>hEvents (Send (Sender A',Proxy,Receiver I))  = (1,1,0,1)
>hEvents (Send (Sender I,Proxy,Receiver I))   = (1,1,2,2)
>hEvents (Send (Sender A',Proxy,Receiver B')) = (1,1,0,0)
>hEvents (Send (Sender I,Proxy,Receiver B'))  = (1,1,2,2)

>dEvents :: Event -> Channel
>dEvents (Send (Sender I,Proxy,Receiver B))  = (1,1,2,2)
>dEvents (Send (Sender I,Proxy,Receiver I))  = (1,1,2,2)
>dEvents (Send (Sender I,Proxy,Receiver B')) = (1,1,2,2)
>dEvents (Send (Sender A,Proxy,Receiver B))  = (1,1,1,2)
>dEvents (Send (Sender A',Proxy,Receiver B)) = (1,1,1,2)
>dEvents (Send (Sender A,Proxy,Receiver I))  = (1,1,2,2)
>dEvents (Send (Sender A',Proxy,Receiver I)) = (1,1,2,2)
>dEvents (Send (Sender A,Proxy,Receiver B')) = (1,1,2,2)
>dEvents (Send (Sender A',Proxy,Receiver B'))= (1,1,2,2)

>trd (_,_,x,_) = x
>fth (_,_,_,x) = x
```