On the Specification of Secure Channels

Christopher Dilloway and Gavin Lowe

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK
{christopher.dilloway,gavin.lowe}@comlab.ox.ac.uk

Abstract. Security architectures often make use of secure transport protocols to protect network messages: the transport protocols provide secure channels between hosts. In this paper we present a hierarchy of specifications for secure channels. We give trace specifications capturing a number of different confidentiality and authentication properties that secure channels might satisfy, and compare their strengths. We give examples of transport layer protocols that we believe satisfy the channel specifications.

1 Introduction

A popular technique for designing a security architecture is to rely on a secure transport layer to protect messages on the network, and provide a secure channel between different hosts. This can simplify the design of the security architecture: the designer can use an off-the-shelf secure transport protocol, such as TLS, to provide secrecy and authentication guarantees; the architecture can then provide additional security guarantees in a higher layer, which we refer to as the application layer.

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. TLS provides strong guarantees; however, it is computationally-expensive, and so in some circumstances, a simpler protocol might suffice.

This layered approach can also simplify the analysis of the architecture. Rather than modelling explicitly the design of the secure transport layer protocol, one can simply model the services it provides, treating it as an abstract secure channel. This results in a simpler model, that concentrates on the application layer. This is the standard approach to analysing layered architectures in other settings. The alternative, of explicitly modelling the functionality of both layers, would lead to unnecessary added complexity.

The aim of this paper, therefore, is to improve our understanding of security guarantees that might be provided by a secure channel. We capture security properties using CSP trace specifications, building on the work of Broadfoot and Lowe [3]. Our formalism will allow 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 the next section we produce a model of a layered network, capturing the events passed between the application layer and the secure transport layer, and between the secure transport layer and the underlying network. In Section 3, we produce a formal specification of confidential channels. Then in Section 4 we consider authentication properties. We start by describing a number of basic building blocks. Not all combinations of the building blocks make sense, and not all combinations are essentially different; we produce a hierarchy containing 7 confidential channels, and 4 non-confidential ones. We then extend this hierarchy by considering session and stream properties, giving a total of 31 different channel properties. We describe a few interesting combinations, and give protocols that we believe satisfy those properties. We summarise and discuss related work in Section 5.

2 An abstract layered network

We formalise our secure channels with respect to an abstract model of the network. Our model uses events at two levels. Most of the events are at the interface between the secure transport layer and the application layer, and describe the application-layer data: these events are enough to capture authentication guarantees. The model also uses events at the interface between the secure transport layer and the underlying network, which describe the network messages: these events are necessary to capture confidentiality properties formally.

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). See Fig. 1.

CSP We will specify channels by giving trace specifications Our specifications should be interpreted in the traces model of CSP; for full details see [11]. A CSP process satisfies a trace predicate if all of its traces satisfy the predicate: P **sat** $R(tr)$ if $\forall tr \in traces(P) \cdot R(tr)$.

A trace is a sequence of events that a CSP process might perform; for example, $\langle a_1, a_2, \ldots, a_n \rangle$ is the trace containing a_1 to a_n in that order; $\langle\rangle$ is the empty trace. If tr and tr' are two finite traces then $tr \frown tr'$ is their concatenation. We write $tr \leqslant tr'$ if tr is a prefix of tr'. We write a **in** tr if the event a occurs in the trace tr.

If c is a channel then $\{| \ c \ |\}$ is the set of events over c. $tr \downarrow c$ is the sequence of data communicated in tr over the channel (or set of channels) c; for example: $\langle a.1, b.1, a.2, b.2, a.3 \rangle \downarrow a = \langle 1, 2, 3 \rangle$.

Describing a channel We consider a secure channel to connect two agents, each playing a particular role.

We assume a set *AgentID* of agent identities; each agent is either *Honest* (i.e., the agent follows the protocol) or *Dishonest* (i.e., the agent is under the intruder's control). We also assume a set *Role* of roles in the protocol, ranged over by R_i, R_j, etc. We define $Agent \mathrel{\widehat{=}} AgentID \times Role$, i.e., particular agents in particular roles. We use A, B, etc., to range over either *AgentID* or *Agent*, as

convenient. We abuse notation by sometimes writing *Honest* for $\mathit{Honest} \times \mathit{Role}$, and similarly for *Dishonest*.

Each type of channel will connect some role R_i to another R_j; we will write $R_i \rightarrow R_j$ for such a channel.

Agents and events The agents, including those under the intruder's control, communicate in sessions, distinguished locally by connection identifiers. A connection identifier can be thought of as a handle to the communication channel: when the protocol agent creates a new channel, a connection identifier will be returned, which the agent will use for all future communication over that channel.

There are three distinct ways in which the intruder can alter and insert messages on the network dishonestly: he can create the messages himself and send them with another agent's identity (i.e. *fake*); he can take an existing message (which was sent by an honest agent) and *redirect* it to a different honest agent; and he can change the identity of the apparent sender of a message (i.e. *hijack* it).[1] We will use the following events to describe these activities, where m ranges over the set $\mathit{Message}_{\mathit{App}}$ 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, which he knows, to agent (B, R_j), in connection c_B; the intruder fakes the message with the identity of honest agent (A, R_i); he may be injecting the message into a pre-existing connection, or causing B to start a new one.

hijack$.(A, R_i).(A', R_i).(B, R_j).c_B.m$**:** the intruder intercepts a previously sent message m and changes the sender from (A, R_i) to (A', R_i) in the message envelope; we will write this event as $\mathit{hijack}.(A, R_i) \rightarrow (A', R_i).(B, R_j).c_B.m$ to highlight its intent.

redirect$.(A, R_i).(B, R_j).(B', R_j).c_{B'}.m$**:** the intruder intercepts a previously sent (or hijacked) message m and changes the receiver (and connection identifier) in the message envelope from (B, R_j) to (B', R_j) so that B' accepts it in connection $c_{B'}$; we will write this event as $\mathit{redirect}.(A, R_i).(B, R_j) \rightarrow (B', R_j).c_{B'}.m$.

For example, if application-layer message m from A to B is encoded as the transport-layer message[2] $A, \{m\}_{PK(B)}$, where $PK(B)$ is B's public key, then a dishonest agent may hijack 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 hijack it by replacing the

---

[1] We use the term hijack to refer to the intruder taking control of a message; if he wishes to redirect it, this is a separate event.

[2] When we give example transport layer protocols, we assume that there are no interactions between the messages of the two layers; we conjecture that a property similar to the disjoint encryption property of [6] is enough to ensure this.

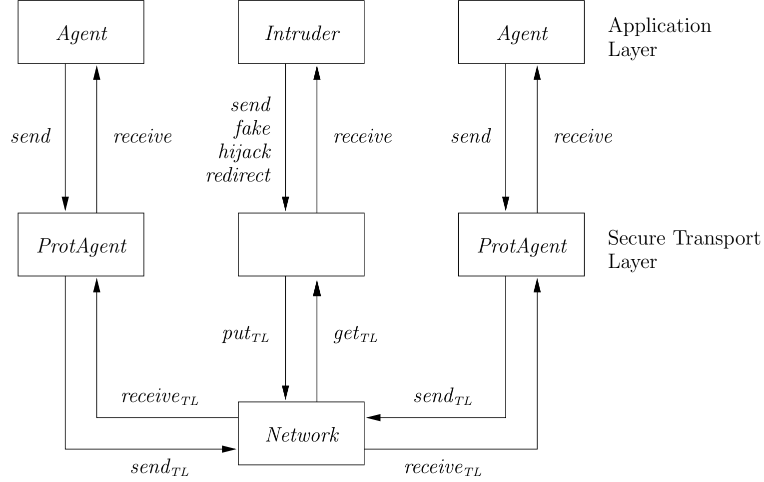


Fig. 1. The network model.

signature with his own: he can only hijack it with a dishonest identity. Note that in both the above cases, the intruder could not have used a *fake* event, except in the first case 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 in both the above cases, the intruder could not have used a *fake*.

The intruder may hijack a message and then redirect it in order to change both the sender and the receiver. In order to simplify the specifications, we do not allow a redirected message to be hijacked.

The network We now specify five rules that define our network. The intruder never sends or fakes messages to himself, and never fakes messages with a dishonest identity (as he can perform a send).

$$\begin{aligned} tr \downarrow \{\!|\ & \mathit{send.Dishonest.Connection.Dishonest}, \\ & \mathit{fake.Agent.Connection.Dishonest},\ \mathit{fake.Dishonest}\ |\!\} = \langle\rangle\ . \end{aligned} \tag{1}$$

The intruder can only hijack messages that were previously sent (not faked).

$$\begin{aligned} & \forall A', A, B : \mathit{Agent}; c_B : \mathit{Connection}; m : \mathit{Message}_{App} \cdot \\ & \quad \mathit{hijack}.A' \to A.B.c_B.m \ \textbf{in}\ tr \Rightarrow \\ & \qquad \exists c_{A'} : \mathit{Connection} \cdot \mathit{send}.A'.c_{A'}.B.m \ \textbf{in}\ tr\ . \end{aligned} \tag{2}$$

The intruder can only redirect messages that were previously sent or hijacked.

$$\begin{aligned}&\forall A, B', B : \mathit{Agent}; c_B : \mathit{Connection}; m : \mathit{Message}_{\mathit{App}} \cdot \\ &\quad \mathit{redirect}.A.B' \rightarrow B.c_B.m \ \mathbf{in}\ tr \Rightarrow \\ &\qquad \exists c_A : \mathit{Connection} \cdot \mathit{send}.A.c_A.B'.m \ \mathbf{in}\ tr \vee \\ &\qquad \exists A' : \mathit{Agent}; c_{B'} : \mathit{Connection} \cdot \mathit{hijack}.A' \rightarrow A.B'.c_{B'}.m \ \mathbf{in}\ tr\ .\end{aligned} \tag{3}$$

In order to define the intruder's capabilities, we require a means to describe exactly what the intruder knows. In the next section we will define a function $\mathit{IntruderKnows} : \mathit{Trace} \rightarrow \mathbb{P}(\mathit{Message}_{\mathit{App}})$ such that $\mathit{IntruderKnows}(tr)$ gives the set of messages that the intruder knows (and so can send) after tr. We limit the intruder's actions in the application layer: he can only send or fake messages that he knows.

$$\begin{aligned}&\forall I : \mathit{Dishonest}; c_I : \mathit{Connection}; B : \mathit{Honest}; tr' : \mathit{Trace}; m : \mathit{Message}_{\mathit{App}} \cdot \\ &\quad tr' \frown \langle \mathit{send}.I.c_I.B.m \rangle \leqslant tr \Rightarrow m \in \mathit{IntruderKnows}(tr') \wedge \\ &\forall A, B : \mathit{Honest}; c_B : \mathit{Connection}; tr' : \mathit{Trace}; m : \mathit{Message}_{\mathit{App}} \cdot \\ &\quad tr' \frown \langle \mathit{fake}.A.B.c_B.m \rangle \leqslant tr \Rightarrow m \in \mathit{IntruderKnows}(tr')\ .\end{aligned} \tag{4}$$

No agent may receive a message that was not previously sent, faked, hijacked or redirected to them.

$$\begin{aligned}&\forall B : \mathit{Honest}; c_B : \mathit{Connection}; A : \mathit{Agent}; m : \mathit{Message}_{\mathit{App}} \cdot \\ &\quad \mathit{receive}.B.c_B.A.m \ \mathbf{in}\ tr \Rightarrow \exists A', B' : \mathit{Agent}; c_A : \mathit{Session} \cdot \\ &\qquad \mathit{send}.A.c_A.B.m \ \mathbf{in}\ tr \vee \mathit{fake}.A.B.c_B.m \ \mathbf{in}\ tr \vee \\ &\qquad \mathit{hijack}.A' \rightarrow A.B.c_B.m \ \mathbf{in}\ tr \vee \mathit{redirect}.A.B' \rightarrow B.c_B.m \ \mathbf{in}\ tr\ .\end{aligned} \tag{5}$$

Relating the abstract network to a concrete network We specify our channels as restrictions on the activity allowed in the application layer. In order to relate our results to a concrete model, we must show how the application-layer events correspond to transport-layer events. See Fig. 1.

When an agent sends a message in the application layer (i.e. performs a *send* event), the protocol agent creates a corresponding send_{TL} event in the transport layer. The network then generates a $\mathit{receive}_{TL}$ event for the recipient (unless the intruder interferes with the message first), which causes the recipient's protocol agent to create a *receive* event in the application layer.

The intruder does not perform send_{TL} or $\mathit{receive}_{TL}$ events; he either adds transport-layer messages to the network (put_{TL}) or removes them from it (get_{TL}). The events the intruder performs in the application layer (*send*, *receive*, *fake*, *hijack* and *redirect*) define the intruder's high-level strategy; the transport-layer events define the implementation of that strategy. For example, in order to hijack or to redirect a message, the intruder will get the transport-layer message, modify it, and then put it back.

Specifying channels For any of our authentication specifications, P, and a channel $R_i \rightarrow R_j$, when we write $P(R_i \rightarrow R_j)$, we mean that:[3]

$$\mathit{System} \setminus \{\!|\ \mathit{send}_{TL}, \mathit{receive}_{TL}, \mathit{put}_{TL}, \mathit{get}_{TL}\ |\!\} \ \mathbf{sat}\ P(R_i \rightarrow R_j)\ ; \tag{6}$$

[3] $P \setminus S$ is the process that behaves like P with all events from the set S hidden.

in other words the specifications only refer to the high-level events.

In order to prove that a particular transport layer protocol really does satisfy a channel specification, one would have to define a protocol agent, translating between application-layer and transport-layer messages, and prove that all traces of the resulting system satisfy the trace specification.

Note that if we have two channel specifications P and Q such that $P \Rightarrow Q$, then a channel that satisfies P can be used anywhere a channel that satisfies Q can be used. In Section 4 we will see some pairs of channels that are not equivalent as predicates, but which can simulate one another; we collapse such pairs.

3 Confidential 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 $C(R_i \to R_j)$).

The *IntruderKnows* function is then defined so that the intruder only learns messages that are sent on non-confidential channels, or that are sent to him:

$$IntruderKnows(tr) \mathrel{\widehat{=}} \{m \mid \begin{pmatrix} IIK \cup SentToIntruder(tr) \cup \\ SentOnNonConfidential(tr) \end{pmatrix} \vdash m\} \ . \qquad (7)$$

$IIK \subseteq Message$ is the intruder's initial knowledge: the set of messages he knows initially. *SentToIntruder* gives the set of messages sent by honest agents to dishonest agents. *SentOnNonConfidential* gives the set of messages sent between agents on non-confidential channels. $\vdash$ is the standard deduction relation, describing how the intruder can deduce new messages from messages he knows, e.g. by decrypting with keys he knows; for details see e.g. [8].

We specify confidential channels by specifying that $IntruderKnows(tr)$, as above, does indeed capture what the intruder would know after trace tr. The messages the intruder will know after observing a trace are those that can be deduced from his initial knowledge and the messages sent on the network:

$$\begin{array}{l} IntruderKnows_{TL}(tr) \mathrel{\widehat{=}} \\ \quad \{m \mid IIK \cup \{m' \mid send_{TL}.Agent.Connection.Agent.m' \ \mathbf{in} \ tr\} \vdash m\} \ . \end{array} \qquad (8)$$

The confidential channels must protect the confidentiality of the messages sent on them: in other words, although the intruder can see the transport-layer messages that are sent on the network, he ought not to be able to deduce the application-layer messages within them. We specify confidential channels as follows: for all traces tr of the system:

$$IntruderKnows_{TL}(tr) \cap Message_{App} = IntruderKnows(tr) \ . \qquad (9)$$

4 Authenticated channels

In this section we specify authenticated channels by specifying under what circumstances an agent may perform a particular *receive* event.

Building blocks for authenticated channels There are three dishonest activities the intruder can perform: faking, hijacking and redirecting. As shown by the examples in Section 2, with some transport protocols the latter two can only be performed using his own identity. We specify our channels by placing restrictions on when he can perform these events. These restrictions are the building blocks that we use to construct more interesting properties. Below we use $\hat{R}_i$ as a shorthand for $AgentID \times \{R_i\}$.

Definition 1 (No faking). *If $NF(R_i \to R_j)$ then the intruder cannot fake messages on the channel:*

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

Definition 2 (No-honest-hijacking). *If $NH^-(R_i \to R_j)$ then the intruder cannot hijack messages using an honest identity:*

$$NH^-(R_i \to R_j)(tr) \mathrel{\widehat{=}} tr \downarrow \{\!|\, hijack.\hat{R}_i \!\to\! (Honest, R_i).\hat{R}_j \,|\!\} = \langle\rangle \; .$$

Definition 3 (No-hijacking). *If $NH(R_i \to R_j)$ then the intruder cannot hijack any messages:*

$$NH(R_i \to R_j)(tr) \mathrel{\widehat{=}} tr \downarrow \{\!|\, hijack.\hat{R}_i.\hat{R}_j \,|\!\} = \langle\rangle \; .$$

Definition 4 (No-honest-redirecting). *If $NR^-(R_i \to R_j)$ then the intruder cannot redirect messages that were sent to honest agents:*

$$NR^-(R_i \to R_j)(tr) \mathrel{\widehat{=}} tr \downarrow \{\!|\, redirect.\hat{R}_i.(Honest, R_j) \!\to\! \hat{R}_j \,|\!\} = \langle\rangle \; .$$

Definition 5 (No-redirecting). *If $NR(R_i \to R_j)$ then the intruder cannot redirect any messages:*

$$NR(R_i \to R_j)(tr) \mathrel{\widehat{=}} tr \downarrow \{\!|\, redirect.\hat{R}_i.\hat{R}_j \,|\!\} = \langle\rangle \; .$$

All of the above specifications work by blocking events; when we specify this we do not mean that the intruder cannot generate the application-layer fake, hijack and redirect events on the channels. What we intend is that when the intruder generates such events, he will either be unable to modify the transport-layer messages in order to generate the necessary put_{TL} events, or the honest protocol agents will reject the messages. Any behaviour of the system where the events are generated but then rejected can be simulated by a behaviour where the events are not created. The simplest way to specify these properties is to ban the events.

The hierarchy We now consider how the building blocks can be combined. They are not independent, since no-hijacking implies no-honest-hijacking, and likewise for no-redirecting.

Further, not all combinations are essentially different: certain pairs of combinations allow essentially the same intruder behaviours as one another: each simulates the other. We therefore collapse such combinations.

1. Non-confidential channels that allow faking but which satisfy one of the forms of no-hijacking or no-redirecting can simulate the bottom channel; the intruder can learn messages and fake them to effect a message hijack or redirect. For example, the trace $\langle \mathit{send}.A.c_A.B.m, \mathit{fake}.A.B'.c_{B'}.m\rangle$ simulates a redirection of m from B to B'.
2. Any hijackable channel that prevents faking can simulate a hijackable channel that allows faking: the intruder can send messages with his own identity, and then hijack them; this activity simulates a fake; e.g. $\langle \mathit{send}.I.c_I.B.m, \mathit{hijack}.I \rightarrow A.B.c_B.m\rangle$.
3. Non-confidential channels that satisfy $\mathit{NF} \wedge \mathit{NH}$ can simulate non-confidential channels that specify $\mathit{NF} \wedge \mathit{NH}^-$; the intruder can always learn messages and then send them with his own identity to simulate a dishonest hijack; e.g. $\langle \mathit{send}.A.B.c_B.m, \mathit{send}.I.c_I.B.m\rangle$.
4. Confidential channels that do not satisfy NR^- or NR can simulate non-confidential channels because the intruder can redirect messages sent on them to himself, and so learn the messages.
5. Confidential, fakeable channels that satisfy NR can simulate confidential, fakeable channels that satisfy NR^-; the intruder learns messages that are sent to him, and so can fake them; e.g. $\langle \mathit{send}.A.c_A.I.m, \mathit{fake}.A.B.c_B.m\rangle$.

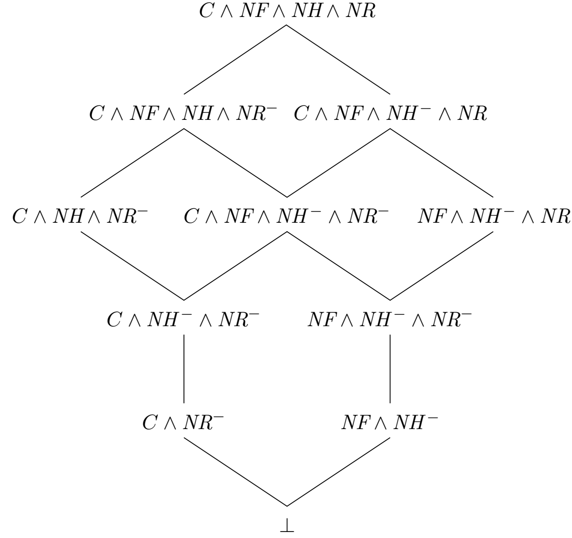
After taking into consideration the collapsing cases described above, we arrive at a hierarchy of 4 non-confidential and 7 confidential channels, shown in Fig. 2 (where several cases collapse to one, the figure gives the weakest specification in each case). We will explain the names in the right-hand column when we discuss those combinations, below.

Stream and session channels We now consider two properties relating different messages in the same connection. These can be combined with the properties of Fig. 2.

In the *record layer* of the TLS protocol [5], the message m is sent as: $\{m, \mathit{hmac}(\mathit{wrtSecret}, \mathit{seqNo}, m)\}_k$. The write secret *wrtSecret* and encryption key k established for any instantiation are almost certainly unique to the session in question, so when an agent receives a message that has been authenticated with a write secret that he recognizes, he can be sure which session it is part of.

Because the sequence number is included in each message he receives, and is authenticated, each agent can be sure that he has not missed any messages, nor received any messages in an order other than that intended by the sender.

This channel provides a strong guarantee to the agents using it: the stream of messages an agent receives is a prefix of the stream of messages sent by the other agent in the session. This property prevents the intruder from rearranging the order in which messages are sent, or inserting or removing messages from a session. The only exception to this is that the intruder can terminate a stream at any point.



Specification	Example	Name
$\perp$	m, A, B	Dolev-Yao
$NF\ NH^-$	$\{m\}_{SK(A)}, B$	Sender authentication
$NF\ NH^-\ NR^-$	$\{h(m, n_A)\}_{SK(A)}, \{n_A\}_{PK(B)}, m$	Responsibility
$NF\ NH^-\ NR$	$\{m, B\}_{SK(A)}$	Strong authentication
$C\ NR^-$	$\{m\}_{PK(B)}, A$	Confidentiality and intent
$C\ NH^-\ NR^-$	$\{m, k\}_{PK(B)}, \{m \oplus k\}_{SK(A)}$	
$C\ NH\ NR^-$	$\{m, A\}_{PK(B)}, A$	Credit
$C\ NF\ NH^-\ NR^-$	$\{h(m, n_A)\}_{SK(A)}, \{m, n_A\}_{PK(B)}$	Responsibility
$C\ NF\ NH^-\ NR$	$\{\{m\}_{PK(B)}\}_{SK(A)}$	
$C\ NF\ NH\ NR^-$	$\{\{m\}_{SK(A)}\}_{PK(B)}, A$	
$C\ NF\ NH\ NR$	$\{\{m, A\}_{PK(B)}\}_{SK(A)}$ or $\{\{m, B\}_{SK(A)}\}_{PK(B)}$	Strong authentication

Fig. 2. The hierarchy of secure channels with example implementations.

Definition 6. *A channel $R_i \to R_j$ is a* stream channel *if*

$$
\begin{aligned}
&St(R_i \to R_j)(tr) \mathrel{\widehat{=}} \forall B : \hat{R}_j; c_B : Connection; A : \hat{R}_i \cdot \\
&\quad \exists A : \hat{R}_i; c_{A'} : Connection; B' : \hat{R}_j \cdot \\
&\qquad tr \downarrow receive.B.c_B.A \leqslant tr \downarrow send.A'.c_{A'}.B' \lor \\
&\quad \exists c_A : Connection \cdot tr \downarrow receive.B.c_B.A \leqslant tr \downarrow fake.A.B.c_B \ .
\end{aligned}
$$

Consider now what happens if we remove the sequence numbers from the TLS record layer messages. The channel now provides a weaker guarantee: the recipient knows that all the messages were sent in the same session, but messages may have been reordered, repeated or cut.

Definition 7. *A channel $R_i \rightarrow R_j$ is a* session channel *if*

$$S(R_i \rightarrow R_j)(tr) \mathrel{\widehat{=}} \forall B : \hat{R}_j; c_B : \mathit{Connection}; A : \hat{R}_i \cdot \\ \exists A' : \hat{R}_i; c_{A'} : \mathit{Connection}; B' : \hat{R}_j \cdot \forall m : \mathit{Message}_{\mathit{App}} \cdot \\ \mathit{receive}.B.c_B.A.m \textbf{ in } tr \Rightarrow \mathit{send}.A'.c_{A'}.B'.m \textbf{ in } tr \vee \\ \forall m : \mathit{Message}_{\mathit{App}} \cdot \mathit{receive}.B.c_B.A.m \textbf{ in } tr \Rightarrow \mathit{fake}.A.B.c_B.m \textbf{ in } tr\,.$$

Each of the properties from Fig. 2 except the bottom one can be strengthened to give a session property by including a session identifier in each message. These can be strengthened further to give a stream property by including a message number, as in TLS. This gives a total of 31 different channels.

Sender authentication The first question we might ask when an agent B receives a message, purportedly from A, is: can he be sure that A really sent the message? In other words: at some point in the past, did A send that message to someone, not necessarily B. We certainly do not want this condition to be met by a $\mathit{fake}.A$ or a $\mathit{hijack}.A' \rightarrow A$ event: we want to guarantee the existence of a $\mathit{send}.A$ event for that message, sometime in the past. However, we shouldn't discount the possibility that A sent a message that the intruder redirected.

Definition 8 (Sender authentication). *The channel $R_i \rightarrow R_j$ provides sender authentication if $NF(R_i \rightarrow R_j) \wedge NH(R_i \rightarrow R_j)$.*

An obvious way to implement this property is for agents to sign messages they send with their secret key, $\{m\}_{SK(A)}$. The signature does not contain the intended recipient's identity, so a channel implemented in this way is redirectable. The intruder cannot fake messages on this channel, nor hijack messages sent by other agents so that they appear to have been sent by A, because he does not know A's secret key. He can, however, learn the message, sign it himself and then send it using his own identity (note that this is a send rather that a hijack); as noted above (item 3), any non-confidential channel that satisfies $NF \wedge NH$ can simulate a non-confidential channel that satisfies $NF \wedge NH^-$ in this way.

With unilateral TLS (i.e. the standard web model), the client is not authenticated to the server. The channel from the server to the client provides authentication of the server's identity. But as the client's identity is not verified, this channel is redirectable (in the sense that the messages may be received by someone other than the agent the server intended them for) and does not satisfy confidentiality. We believe this channel satisfies $St \wedge NF \wedge NH$.

Intent We have just seen that when agents sign messages with their secret key, their intent might not be preserved — the intruder can redirect their messages to whomever he likes. We now specify a channel that provides a guarantee of (the original sender's) intent: whenever B receives a message, he knows that the agent who originally sent it intended him to receive it. On these channels we forbid redirection (as this would allow the intruder to change the recipient so that the sender's intent is not preserved), but we allow faking and hijacking.

Definition 9 (Intent). *The channel $R_i \rightarrow R_j$ provides a guarantee of intent if $NR(R_i \rightarrow R_j)$.*

The easiest way to design a channel that provides a guarantee of intent is to encrypt messages with the intended recipient's public key. We have already used this method as the most obvious implementation of a secret channel.

Recall that non-confidential, non-redirectable, fakeable channels can simulate message redirection by learning messages and faking them (item 1). We therefore always combine intent with confidentiality or non-fakeability. Further, fakeable, confidential channels that satisfy NR can simulate fakeable, confidential channels that satisfy NR^-, because the intruder learns messages that are sent to him, and so can fake them to 'redirect' them to another agent (item 5).

With unilateral TLS, the channel from the client to the server provides a guarantee of the sender's (the client's) intent, as the client must have verified the server's identity; however it does not provide authentication of the client's identity. We believe this channel satisfies $C \wedge St \wedge NR$.

Strong authentication Strong authentication is the combination of the previous two properties: whenever B receives a message from A, A previously sent that message to B; we note that the specification for strong authentication is the conjunction of the previous two specifications.[4]

Definition 10 (Strong authentication). *The channel* $R_i \rightarrow R_j$ *provides strong authentication if* $NF(R_i \rightarrow R_j) \wedge NH(R_i \rightarrow R_j) \wedge NR(R_i \rightarrow R_j)$.

We can achieve strong authentication by strengthening the protocol given for sender authentication to $\{B, m\}_{SK(A)}$. The intruder cannot change the recipient's identity whilst maintaining A's signature, so this channel is unredirectable; he cannot fake messages on this channel because he does not know A's secret key; and he cannot hijack messages so that they appear to have been sent by an honest agent. (As with sender authentication, he can learn the message and sign it himself; again this is not a hijack.) This channel guarantees that when B receives a message apparently from A, then previously A sent it to B.

We believe that bilateral TLS establishes an authenticated stream in each direction, and so satisfies $C \wedge St \wedge NF \wedge NH \wedge NR$. Such a channel is equivalent to the authenticated channels defined by Broadfoot and Lowe [3].

We note that neither $\{\{m\}_{SK(A)}\}_{PK(B)}$ nor $\{\{m\}_{PK(B)}\}_{SK(A)}$ provides strong authentication; the former can be redirected when B is dishonest, and the latter hijacked with a dishonest identity; they satisfy, respectively, $C \wedge NF \wedge NH \wedge NR^-$ and $C \wedge NF \wedge NH^- \wedge NR$.

Credit and responsibility In [1], Abadi highlighted two different facets of authentication. When an agent B receives a message m from an authenticated agent A, he could interpret it in two different ways: (1) he might attribute *credit* for the message m to A; for example, if B is running a competition, and m is an entry to the competition, he would give credit for that entry to A; (2) he might believe that the message is supported by A's authority, and so assign *responsibility* for it to A; for example, if m is a request to delete a file, then his decision will depend on whether or not A has the authority to delete the file.

[4] By analogy with [9], we sometimes refer to sender authentication as *weak authentication*, and (strong) authentication as *authentication*.

Abadi argued that these two interpretations of authentication are not the same, and that protocol designers tend not to state which form of authentication their protocols provide: in many cases protocols will offer one, but not the other.

A non-hijackable and non-redirectable confidential channel is suitable for giving credit. The intruder can fake messages on these channels, but in doing so he only gives another agent credit for his messages.

Definition 11 (Credit). *The channel $R_i \rightarrow R_j$ can be used to give credit if $C(R_i \rightarrow R_j) \wedge NH(R_i \rightarrow R_j) \wedge NR^{-}(R_i \rightarrow R_j)$.*

Abadi gives the following example of a protocol suitable for assigning credit: $\{A, k\}_{PK(B)}, \{m\}_k$. When B receives this message he knows that he can give credit for m to the person who encrypted the key k; however he cannot be sure that it was really A who sent the message. So while the intruder can fake messages on this channel, he will only be giving credit to someone else, rather than claiming it for himself.

A non-fakeable channel with no-honest-hijacking and no-redirecting from honest agents is suitable for authentication protocols where responsibility is claimed.

Definition 12 (Responsibility). *The channel $R_i \rightarrow R_j$ can be used to assign responsibility if $NF(R_i \rightarrow R_j) \wedge NH^{-}(R_i \rightarrow R_j) \wedge NR^{-}(R_i \rightarrow R_j)$.*

In some circumstances, one might wish to strengthen such a channel so that it also provides intent (i.e. $NF \wedge NH^{-} \wedge NR$), to ensure that the correct agent assigns the responsibility.

5 Conclusions and related work

In this paper we have examined a hierarchy of secure channel specifications. We illustrated these channel specifications via example protocols that might implement them, but we have not proven that the implementations are correct. It is clear that the hierarchy presented in this document is not complete: there are other properties (e.g. recentness, non-repudiation) that channels can provide that we have not accounted for in our specifications.

The channel specifications on their own serve as an interesting exploration of the sort of protection that might be afforded by a transport layer. However, they really come into their own when they can be used to analyse security protocols that use secure transport layers. The approach to analysing such protocols that other researchers have taken is to model the transport layer protocol first, and then to model the security protocol being run on top of that; see, e.g. [7]. We propose to analyse security protocols that use secure transport layers using Casper [10] and FDR [11]; to do so, we will build CSP models capturing the services provided by secure channels. This will make analysis of layered protocols no more difficult than a standard Casper analysis. We expect to find suitable example protocols in grid and web architectures, and in studying delegation.

Delegation protocols provide an interesting area for further extensions to the model. In many delegation protocols security credentials are established in the application layer, and then used in the transport layer. This crossing of layers is not something our current model can represent, as we assume that application layer keys and transport layer keys are disjoint. There may also be other classes of security protocol in which data values established in one layer are used in the other, so it would be useful if our model could be extended to enable us to study these.

We briefly discuss how our approach to specifying secure channels compares with that taken by other authors.

In [3], Broadfoot and Lowe specify a form of secrecy that is equivalent to our network with every channel being non-hijackable and confidential. The difference between our definition of confidentiality and that in [3] is that we allow the intruder to change the identity of the sender of a message. In Broadfoot and Lowe's model, the intruder does not possess this capability (he must intercept the message, learn the content, and then recreate it: he cannot merely change the identity of the sender), so their definition ought to be compared to, and is equivalent to, non-hijackable confidential channels. Broadfoot and Lowe also specify a single form of authenticated channel which is equivalent to an authenticated stream channel.

Creese et al. [4] have developed the notion of *empirical channels*, and adapted the traditional attack model they use when analysing protocols in order to study security protocols for pervasive computing. They have a network model comprising traditional, high-bandwidth digital communications channels, and empirical, low-bandwidth and human-oriented, channels. The empirical channels are used for non-traditional forms of communication, which often seem necessary for applications in pervasive computing: for example, two humans comparing a code printed on each of their laptop screens, or a human entering a code on a printer's keypad. Over such channels, they specify any combination of the following restrictions on the intruder: *No spoofing:* the attacker cannot spoof messages on this channel: this corresponds to an unfakeable channel;[5] *No over-hearing:* the attacker cannot overhear messages sent on this channel: this is equivalent to a confidential channel; *No blocking:* the attacker cannot block messages on this channel. We do not have an equivalent to the no blocking channel, because on a traditional network, where the intruder is assumed to be in control of all message flows, we do not see how this anti-denial-of-service property could be realised; on the empirical channels suggested in [4] (such as a human entering a number into a keypad), it is easier to see how this would be possible.

Acknowledgements. We would like to thank Allaa Kamil for many useful discussions. This work is partially funded by the US Office of Naval Research.

[5] Creese et al. actually allow two different models of this channel: one that allows redirecting, and one that doesn't.

References

1. M. Abadi. Two facets of authentication. In *11th IEEE Computer Security Foundations Workshop*, pages 27–32, 1998.
2. M. Bellare and P. Rogaway. Optimal asymmetric encryption — how to encrypt with RSA. *Eurocrypt*, 94:92–111, 1995.
3. P. Broadfoot and G. Lowe. On distributed security transactions that use secure transport protocols. In *16th IEEE Computer Security Foundations Workshop*, 2003.
4. S. Creese, M. Goldsmith, R. Harrison, A. W. Roscoe, P. Whittaker, and I. Zakiuddin. Exploiting empirical engagement in authentication protocol design. *Lecture notes in computer science*, 3450:119–133, 2005.
5. T. Dierks and C. Allen. The TLS protocol version 1.0, 1999.
6. J. D. Guttman and F. J. Thayer Fábrega. Protocol independence through disjoint encryption. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 24–34, 2000.
7. S. M. Hansen, J. Skriver, and H. R. Nielson. Using static analysis to validate the SAML Single Sign-On Protocol. In *WITS*, 2005.
8. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.
9. G. Lowe. A hierarchy of authentication specifications. In *10th IEEE Computer Security Foundations Workshop*, 1997.
10. G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.
11. A.W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.