Specifying Secure Transport Layers

Christopher Dilloway
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford, OX1 3QD, UK
christopher.dilloway@comlab.ox.ac.uk

Gavin Lowe
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford, OX1 3QD, UK
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.

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. [9, 12, 13]. 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 secure channels. We capture security properties using CSP-style 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 Section 1 we formalise an abstract model of a layered network, and relate it to a concrete network. We describe the sets of valid traces that our network accepts, and we provide a framework for specifying secure channels.

In Section 2 we describe how we flag confidential channels in a system, and define the properties of a confidential channel in terms of the relation between the intruder's knowledge in our abstract model and the intruder's knowledge in the concrete model.

In Section 3 we define the building blocks we use to create our hierarchy. These building blocks progressively disallow different aspects of the intruder's behaviour, and can be combined to create different channels. Not all combinations are distinct: in many cases, several different compositions of the building blocks will allow essentially the same behaviour (they simulate one another); we collapse such cases, and reach a hierarchy of eleven secure channels. In Section 4 we consider several of the secure channels from the hierarchy in more detail, and relate them to real-world secure transport protocols.

In Section 5 we consider channel specifications that tie different messages into a single connection. We specify a *session* property that binds messages into a single session, and a stronger *stream* property that not only ensures that messages are not moved from one session to another, but also guarantees that the order messages are received in is the same as that in which they were sent.

In Section 6 we define a simulation relation on systems of secure channels, based on the traces of specifications as they are viewed by the honest agents. Specification $Spec_1$ simulates specification $Spec_2$ if $Spec_1$ allows at least as many traces as viewed by honest agents. We use the simulation relation to define an equivalence relation. In Section 7 we use the equivalence relation to prove the equiva-

lence of an alternative form for each of our channel specifications. Each alternative form describes the necessary behaviour that must precede a receive event, rather than blocking the intruder's behaviour.

Finally, in Section 8 we conclude, and discuss alternative approaches to specifying secure channels, and several pieces of related work.

1. Channels, formally

In this section we formalize our model of an abstract network and its relation to a concrete network. We use CSP-style trace notation: see Appendix A. The abstract network is defined 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.

Our 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. Our model uses entities at two interfaces: between the application layer and the secure transport layer, and between the secure transport layer and the underlying network. The application layer is the layer in which agents establish channels, and send and receive messages. 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 Figure 1.

Most of the events are at the interface between the application layer and the secure transport 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.

Describing a channel We assume a set *Identity* of agent identities. Each identity is either considered *Honest* (i.e. the agent follows the application-layer protocols) or *Dishonest* (i.e. the agent is under the intruder's control).

We also assume a set *Role* of roles in the application-layer protocols, ranged over by R_i, R_j, etc. Each role in an application protocol will exchange a series of messages with some of the other roles in the protocol. We assume that the roles used by different protocols are distinct.

An *Agent* is an identity taking a role: *Agent* $\widehat{=}$ *Identity* $\times$ *Role*. We use A, A', B, B', etc., to range over either *Identity* or *Agent*, as convenient; we use I as a dishonest identity (or agent). We abuse notation by sometimes writing *Honest* for *Honest* $\times$ *Role*, and similarly for *Dishonest*. We write $\hat{R}_i$ for *Identity* $\times$ R_i.

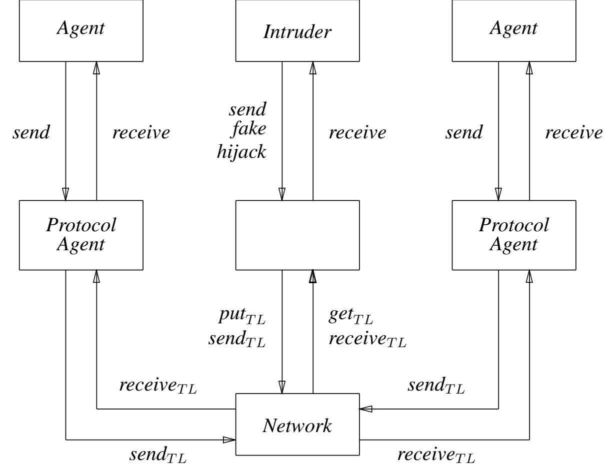


Figure 1. The concrete and abstract levels of a system.

A secure channel connects two agents, each playing a particular role: a channel is an ordered pair of roles $(R_i, R_j) \in$ *Role* $\times$ *Role*. We write $R_i \rightarrow R_j$ for the channel (R_i, R_j) as this emphasises the difference between the sending and the receiving roles.

We treat encryption formally. All messages are drawn from the message space, *Message*. This is a set of symbols built from basic types (such as identities, nonces and timestamps) by operations such as concatenation and encryption. 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; i.e.

$$X \subseteq X' \Rightarrow \{m \mid X \vdash m\} \subseteq \{m \mid X' \vdash m\}\,,$$
$$X \vdash m \wedge X \cup \{m\} \vdash m' \Rightarrow X \vdash m'\,.$$

Often in our examples we use the deduction rules from [11] which model Dolev-Yao style symbolic encryption: 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. However, these rules can be modified, corresponding to other models of encryption, as required; the results in this paper hold for any deduction relation that satisfies the two properties above.

We assume that the intruder has some initial knowledge *IIK* $\subseteq$ *Message*. He may use this knowledge and messages he overhears on the network to generate new messages and facts. We restrict the intruder's behaviour so that he can only send messages that can be deduced from his initial knowledge and what he has overheard.

The message space is partitioned into two sets: application-layer messages ($Message_{App}$) and transport-layer messages ($Message_{TL}$). We assume that there are no interac-

tions between the messages of the two layers; in particular, when we give example transport-layer protocols we assume that the messages we describe could not be confused for application-layer messages. We conjecture that a property similar to the disjoint encryption property of [5] is sufficient to ensure this.

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 communication over that channel. We use c_A, etc., to range over *Connection*.

We use the following events, 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 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 connnection, or causing B to start a new one. In order to fake a message, the intruder must be able to choose the message from those he knows.

hijack$.(A, R_i).(A', R_i).(B, R_j).(B', R_j).c_{B'}.m$: 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 $c_{B'}$; we write this event as *hijack*$.(A, R_i) \rightarrow (A', R_i).(B, R_j) \rightarrow (B', R_j).c_{B'}.m$ to highlight its intent.

The *hijack* event can be used by the intruder in four different ways:

- To *replay* a previously-sent message, the intruder either chooses an existing connection or initialises a new one, and causes the recipient to receive the message in that connection; we abbreviate the event and write *hijack*$.(A, R_i).(B, R_j).c_B.m$;
- To *re-ascribe*[1] a message, the intruder changes the sender's identity and chooses a connection for the recipient to receive it in; we abbreviate the event and write *hijack*$.(A, R_i) \rightarrow (A', R_i).(B, R_j).c_B.m$;
- To *redirect* a message, the intruder changes the identity of the recipient and chooses a connection for the new recipient to receive it in; we abbreviate the event and write *hijack*$.(A, R_i).(B, R_j) \rightarrow (B', R_j).c_{B'}.m$;
- To *re-ascribe and redirect* a message the intruder changes both identities and chooses a connection for the new recipient to receive the message in.

[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.

For example, if application layer message m from A to B is encoded as the transport layer message $A, \{m\}_{PK(B)}$, where $\mathit{PK}(B)$ 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 $\mathit{SK}(A)$ 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 $\mathit{SK}(B)$: 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.

An abstract network We now specify four rules that define the application-layer behaviour accepted by our networks. We are not yet trying to capture channel properties; rather, we are defining some sanity conditions in order to remove artificial and irrelevant behaviour from our networks.

The intruder never sends or fakes messages to himself, never fakes messages with a dishonest identity (as he can perform a send), and never redirects a message sent to himself and re-ascribes it with his own identity (as he can perform a receive and a send).

$$\begin{aligned}\mathcal{N}_1(tr) \mathrel{\widehat{=}} tr \downarrow \{|\ & \mathit{send.Dishonest.Connection.Dishonest},\\ & \mathit{hijack.Agent} \rightarrow \mathit{Dishonest.Dishonest} \rightarrow \mathit{Agent},\\ & \mathit{fake.Agent.Dishonest}, \mathit{fake.Dishonest}\ |\} = \langle\rangle\,.\end{aligned}$$

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

$$\begin{aligned}\mathcal{N}_2(tr) \mathrel{\widehat{=}} & \forall A, A', B, B', c_{B'}, m \cdot\\ & \mathit{hijack}.A \rightarrow A'.B \rightarrow B'.c_{B'}.m \textbf{ in } tr \Rightarrow\\ & \exists c_A \cdot \mathit{send}.A.c_A.B.m \textbf{ in } tr\,.\end{aligned}$$

In order to define the intruder's capabilities, we require a means to describe exactly what the intruder knows. In the next section we define a function:

$$\mathit{IntruderKnows}_{IIK} : \mathit{Trace} \rightarrow \mathbb{P}(\mathit{Message}_{\mathit{App}})$$

such that $\mathit{IntruderKnows}_{IIK}(tr)$ gives the set of messages that the intruder knows (and so can send) after *tr*, assuming that his initial knowledge is *IIK*. We limit the intruder's

actions in the application layer: he can only send or fake messages that he knows.

$$\begin{aligned}
&\mathcal{N}_{3,IIK}(tr) \mathrel{\widehat=} \\
&\quad \forall I : \mathit{Dishonest}, c_I, B, tr', m \cdot \\
&\qquad tr' \frown \langle \mathit{send}.I.c_I.B.m \rangle \leqslant tr \Rightarrow \\
&\qquad\quad m \in \mathit{IntruderKnows}_{IIK}(tr') \wedge \\
&\quad \forall A, B, c_B, tr', m \cdot tr' \frown \langle \mathit{fake}.A.B.c_B.m \rangle \leqslant tr \Rightarrow \\
&\qquad m \in \mathit{IntruderKnows}_{IIK}(tr') \,.
\end{aligned}$$

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

$$\begin{aligned}
&\mathcal{N}_4(tr) \mathrel{\widehat=} \forall B, c_B, A, m \cdot \\
&\quad \mathit{receive}.B.c_B.A.m \ \mathbf{in}\ tr \Rightarrow \exists A', B', c_A \cdot \\
&\qquad \mathit{send}.A.c_A.B.m \ \mathbf{in}\ tr \vee \mathit{fake}.A.B.c_B.m \ \mathbf{in}\ tr \vee \\
&\qquad \mathit{hijack}.A' \rightarrow A.B' \rightarrow B.c_B.m \ \mathbf{in}\ tr \,.
\end{aligned}$$

Relating the abstraction to a concrete network

When an agent sends a message in the application layer (i.e. performs a *send* event), their 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 hijacks the message first), which causes the recipient's protocol agent to perform a *receive* event in the application layer.

We assume the existence of a partial, symbolic decoding function $\mathcal{D} : (\mathit{Agent} \times \mathit{Agent}) \times \mathit{Connection} \times \mathit{Trace} \nrightarrow \mathit{Trace}$ 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. There is not necessarily a 1-1 relationship between application-layer and transport-layer messages: some channels may have an initial key-establishment phase, or may send several transport-layer messages for each application-layer message, or aggregate several application-layer messages into a single transport-layer message. This decoding function gives the trace of application-layer events that would result from undoing encryption, validating signatures and performing other functions necessary for the implementation of the secure channel in use.

The decoding function is clearly dependent upon the implementation of the secure channel from A to B, and also upon the security parameters (e.g. keys) established for any particular connection. Its result, for a given connection, depends only on the transport layer messages sent and received on that connection, and satisfies an obvious prefix property:

$$\begin{aligned}
&\forall A : \hat{R}_i, B : \hat{R}_j, c_A, tr, tr' \cdot \\
&\quad tr' \downarrow \{send_{TL}.A.c_A.B, receive_{TL}.A.c_A.B\} \leqslant \\
&\qquad tr \downarrow \{send_{TL}.A.c_A.B, receive_{TL}.A.c_A.B\} \Rightarrow \\
&\qquad\quad \mathcal{D}(A \rightarrow B)(c_A)(tr') \leqslant \mathcal{D}(A \rightarrow B)(c_A)(tr) \,.
\end{aligned}$$

The protocol agents faithfully translate the transport-layer events into application-layer events, possibly with some buffering, so the decoding of the transport-layer messages gives a prefix of the corresponding application-layer $send$s, and a postfix of the corresponding application-layer $receive$s.

$$\begin{aligned}
&\mathcal{A}_1(tr) \mathrel{\widehat=} \forall A : (\mathit{Honest}, R_i), c_A, B : \hat{R}_j \cdot \\
&\quad (\mathcal{D}(A \rightarrow B)(c_A)(tr)) \downarrow \mathit{send}.A.c_A.B \leqslant \\
&\qquad tr \downarrow \mathit{send}.A.c_A.B \,, \\
&\mathcal{A}_2(tr) \mathrel{\widehat=} \forall B : (\mathit{Honest}, R_j), c_B, A : \hat{R}_i \cdot \\
&\quad tr \downarrow \mathit{receive}.B.c_B.A \leqslant \\
&\qquad (\mathcal{D}(A \rightarrow B)(c_B)(tr)) \downarrow \mathit{receive}.B.c_B.A \,.
\end{aligned}$$

The intruder has additional capabilities: as well as performing send_{TL} or $\mathit{receive}_{TL}$ events he can add transport layer messages to the network (put_{TL}) or remove them from it (get_{TL}). The events the intruder performs in the application layer (*send*, *receive*, *fake*, and *hijack*) define his high-level strategy; the transport-layer events define the implementation of that strategy. For example, in order to hijack a message, the intruder will *get* the transport layer message, modify it, and then *put* it back.

We do not directly specify a formal relationship between the intruder's application-layer and transport-layer events, since the intruder is not forced to follow the protocol. In general however, 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.

Specifying channels We specify channels by giving trace specifications. 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.

The set of valid system traces is the (prefix-closed) set of traces that satisfy properties $\mathcal{N}_1$–$\mathcal{N}_4$ and $\mathcal{A}_1$–$\mathcal{A}_2$.

$$\begin{aligned}
&\mathit{ValidSystemTraces}_{IIK} \mathrel{\widehat=} \{tr \in \{| \mathit{send}_{TL}, \mathit{receive}_{TL}, \\
&\quad \mathit{put}_{TL}, \mathit{get}_{TL}, \mathit{send}, \mathit{receive}, \mathit{fake}, \mathit{hijack} |\}^* \mid \\
&\qquad \forall tr' \leqslant tr \cdot \\
&\qquad\quad \mathcal{N}_1(tr') \wedge \mathcal{N}_2(tr') \wedge \mathcal{N}_{3,IIK}(tr') \wedge \mathcal{N}_4(tr') \wedge \\
&\qquad\quad \mathcal{A}_1(tr') \wedge \mathcal{A}_2(tr')\} \,.
\end{aligned}$$

A channel specification is a predicate over traces:

$$\mathit{ChannelSpec} : \mathit{ValidSystemTraces}_{IIK} \rightarrow \mathbb{B} \,.$$

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

$$\begin{aligned}
&\mathit{traces}_{IIK}(\mathit{ChannelSpec}) = \\
&\quad \{tr \in \mathit{ValidSystemTraces}_{IIK} \mid \\
&\qquad \forall tr' \leqslant tr \cdot \mathit{ChannelSpec}(tr')\} \,.
\end{aligned}$$

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 and, in this case, $\mathit{traces}(P) \subseteq \mathit{traces}(Q)$. In Section 3 we will see some pairs of channels that are not equivalent as predicates, but which can simulate one another; we collapse such pairs.

2. 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 $C(R_i \rightarrow R_j)$).

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:

$$\mathit{IntruderKnows}_{IIK}(tr) \mathrel{\widehat{=}} \{m \mid (\mathit{SentToIntruder}(tr) \cup \mathit{SentOnNonConfidential}(tr) \cup \mathit{IIK}) \vdash m\} .$$

SentToIntruder gives the set of messages sent by honest agents to dishonest agents, and *SentOnNonConfidential* gives the set of messages sent between agents on non-confidential channels.

The traces of a confidential channel specification depend on the intruder's initial knowledge. When we claim that a secure transport layer is confidential, we make that claim subject to a restriction on the intruder's initial knowledge (usually that the intruder's initial knowledge does not contain the honest agents' secret keys).

We specify confidential channels by requiring that the $\mathit{IntruderKnows}(tr)$ function 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:

$$\mathit{IntruderKnows}_{TL,IIK}(tr) \mathrel{\widehat{=}} \{m \mid \{m' \mid \exists A, B, c_A \cdot \mathit{send}_{TL}.A.c_A.B.m' \textbf{ in } tr\} \cup \mathit{IIK} \vdash m\} ;$$

so we require that:

$$\forall tr \in \mathit{traces}_{IIK}(\mathit{ChannelSpec}) \cdot \mathit{IntruderKnows}_{IIK}(tr) = \mathit{IntruderKnows}_{TL,IIK}(tr) \cap \mathit{Message}_{\mathit{App}} .$$

3. Authenticated 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. As the examples in Section 1 show, 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. The restrictions below are the building blocks that we use to construct more interesting properties.

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

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

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

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

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

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

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

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

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

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

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 and hijack 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 intruder can use a hijack event to force an honest agent to receive a message in a particular connection without changing either of the identities associated with the message. This activity is not blocked by any of the properties above. In particular, the intruder can force an agent to receive a message more times than it is sent, i.e. to replay a message. We do not specify a no-replaying property in the building blocks because we do not wish to consider it independently. In Section 5 we give session properties that indirectly prevent replaying, and also properties that provide stronger guarantees.

Combining the building blocks We now consider how the building blocks can be combined. They are not independent, since no-re-ascribing implies no-honest-re-ascribing, and likewise for no-redirecting. Further, not all combinations are essentially different; certain pairs of combinations allow essentially the same intruder behaviours: each simulates the other (see Section 6). We therefore collapse such combinations.

- $\mathcal{C}_1$ Non-confidential channels that allow faking but which satisfy one of the forms of no-re-ascribing or no-redirecting can simulate the bottom channel; the intruder can learn messages and fake them to effect a message re-ascribe or redirect.

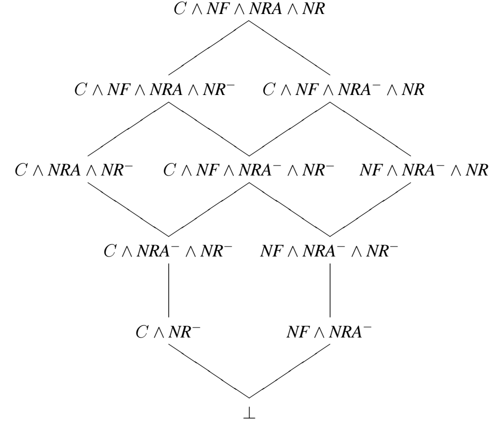
- $\mathcal{C}_2$ Any re-ascribable channel that prevents faking can simulate a re-ascribable channel that allows faking: the intruder can send messages with his own identity and then re-ascribe them; this activity simulates a fake.

- $\mathcal{C}_3$ Non-confidential channels that satisfy $NF \wedge NRA$ can simulate non-confidential channels that satisfy $NF \wedge NRA^-$; the intruder can always learn messages and then send them with his own identity to simulate a dishonest re-ascribe.

- $\mathcal{C}_4$ 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.

- $\mathcal{C}_5$ 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.

After taking these collapsing cases into consideration we arrive at a hierarchy of four non-confidential and seven confidential channels, shown in Figure 2 (where several cases collapse to one, the figure gives the weakest specification in each case). We also give simple example transport protocols that we believe satisfy each of the properties; we explain the names in the right-hand column when we discuss these combinations in the next section.



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

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

4. Some interesting authenticated channels

In this section we examine some of the channels in the hierarchy in more detail, and describe which of these properties we believe are satisfied by some standard secure transport layers.

Sender authentication When an agent *B* receives a message, purportedly from *A*, he might ask whether he can 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*?

Definition 4.1. The channel $R_i \rightarrow R_j$ provides *sender authentication* if $NF(R_i \rightarrow R_j) \wedge NRA(R_i \rightarrow R_j)$.

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 re-ascribe messages sent by other agents so that they appear to have been sent by A, because he does not know A's secret key.

With unilateral TLS (i.e. the standard web model), the channel from the server to the client provides authentication of the server's identity, but not of the client's. This channel is redirectable because the messages may be received

by someone other than the agent the server intended them for, and so does not satisfy confidentiality. We believe this channel satisfies $\mathit{StrongStream} \wedge \mathit{NF} \wedge \mathit{NRA}$.[2]

Intent When agents sign messages with their secret key, their intent might not be preserved — the intruder can redirect their messages to whomever he likes.

Definition 4.2. The channel $R_i \rightarrow R_j$ provides a guarantee of *intent* if $\mathit{NR}(R_i \rightarrow R_j)$.

In other words, the recipient of a message knows that the sender intended him to receive it.

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 confidential channel.

Recall that non-confidential, non-redirectable, fakeable channels can simulate message redirection by learning messages and faking them (collapsing case $\mathcal{C}_1$). 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 (collapsing case $\mathcal{C}_5$).

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 $C \wedge \mathit{StrongStream} \wedge \mathit{NR}$.

Strong authentication We can combine the previous two properties so that whenever *B* receives a message from *A*, *A* previously sent that message to *B*.

Definition 4.3. The channel $R_i \rightarrow R_j$ provides *strong authentication* if it provides sender authentication and intent $\mathit{NF}(R_i \rightarrow R_j) \wedge \mathit{NRA}(R_i \rightarrow R_j) \wedge \mathit{NR}(R_i \rightarrow R_j)$.

We can achieve strong authentication by encoding m as $\{B, m\}_{SK(A)}$. 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 re-ascribe 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 re-ascribe.)

We believe that bilateral TLS establishes an authenticated stream in each direction, and so both channels satisfy $C \wedge \mathit{StrongStream} \wedge \mathit{NF} \wedge \mathit{NRA} \wedge \mathit{NR}$. Such a channel is equivalent to the authenticated channels of Broadfoot and Lowe [3].

[2] *StrongStream* is a session property and is defined in Section 5.

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. 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. Alternatively, 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.

Abadi argued that these 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.

Definition 4.4. The channel $R_i \rightarrow R_j$ can be used to give *credit* if $C(R_i \rightarrow R_j) \wedge \mathit{NRA}(R_i \rightarrow R_j) \wedge \mathit{NR}^-(R_i \rightarrow R_j)$.

The intruder can fake messages on these channels, but in doing so he only gives another agent credit for his messages.

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 did this. 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.

Definition 4.5. The channel $R_i \rightarrow R_j$ can be used to assign *responsibility* if $\mathit{NF}(R_i \rightarrow R_j) \wedge \mathit{NRA}^-(R_i \rightarrow R_j) \wedge \mathit{NR}^-(R_i \rightarrow R_j)$.

The only attack the intruder could perform on such a channel would be to overhear a message, or to claim it as his own. In the latter case, he will either not have the authority required for the message (as in the example of a fileserver and a delete message), or he will be accepting the blame for something. The example given for a strongly authenticated channel would be a suitable implementation of this channel.

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

Guaranteed knowledge Both of the previous channels (credit and responsibility) provide a further property: they guarantee that the apparent sender of a message knew the content of the message. This is important for these channels as an agent should not be able to claim credit for a message that he doesn't know, and no agent should claim responsibility for a message that he does not know.

Fakeable channels cannot provide this property: if the intruder can fake messages with another agent's identity, he can send messages that they have no chance of knowing.

Further, if the intruder can re-ascribe a message to an honest agent then the channel cannot provide guaranteed knowledge. If the intruder can re-ascribe a message to himself (i.e. to a dishonest agent) then the channel can only provide guaranteed knowledge if it is non-confidential.

Definition 4.6. The channel $R_i \rightarrow R_j$ provides a guarantee that the apparent sender of a message knew the message if $\mathit{NF}(R_i \rightarrow R_j) \wedge \mathit{NRA}^-(R_i \rightarrow R_j) \wedge (C(R_i \rightarrow R_j) \Rightarrow \mathit{NRA}(R_i \rightarrow R_j))$.

5. Session and stream channels

The properties described so far allow us to specify channels that provide guarantees for individual messages. In practice it is often necessary to group together different messages that were sent in the same connection into a single *session*. In this section we consider six properties relating different messages in the same connection; they can be combined with the properties of Figure 2.

Given a trace tr we might ask whether it is feasible that the event $\mathit{send}.A.c_A.B.m$ is responsible for the event $\mathit{receive}.B'.c_{B'}.A'.m$, in the sense that if the first event had not happened, the second might not have. If either $A \neq A'$ or $B \neq B'$ there must be a hijack event between the two events in order for the first to be responsible for the second.

$$\begin{aligned}
&\mathit{Feasible}_{tr}(\mathit{send}.A.c_A.B.m, \mathit{receive}.B'.c_{B'}.A'.m) \mathrel{\widehat{=}} \\
&\quad \exists tr', tr'' : \mathit{Trace} \cdot tr' {}^\frown \langle \mathit{send}.A.c_A.B.m \rangle {}^\frown tr'' {}^\frown \\
&\qquad \langle \mathit{receive}.B'.c_{B'}.A'.m \rangle \leqslant tr \wedge \\
&\qquad\quad (A = A' \wedge B = B') \vee \\
&\qquad\quad (\mathit{hijack}.A \rightarrow A'.B \rightarrow B'.c_{B'}.m \ \mathbf{in}\ tr'') \,.
\end{aligned}$$

The hijack event is only defined when the roles played by the new sender and receiver are the same as those played by the old. In order for it to be feasible that a particular send event is responsible for a receive event we assume, implicitly, that the roles of the new and the old sender are the same, and likewise for the receivers.

Any system trace tr induces a relation $\mathcal{R}_{tr}$ (*receives-from*) over the set of connection identifiers in that system: $c_B \mathcal{R}_{tr} c_A$ if *all* of the messages received in the connection c_B could feasibly have been sent in the connection c_A.

We need to be careful about the way we deal with connections that receive faked messages. The *fake* event is an abstraction of the activity that the intruder performs when he fakes messages: he creates a new protocol agent with a false identity, and then uses that protocol agent to establish connections to other agents. He then uses these connections to fake messages. We partition *Connection* into honest and intruder connection identifiers. An honest agent's connection that receives faked messages is related to every intruder connection.

$$\begin{aligned}
&c_B \mathcal{R}_{tr} c_A \mathrel{\widehat{=}} \forall A', B : \mathit{Agent} \cdot \\
&\quad \exists A, B' \cdot \forall m \cdot \mathit{receive}.B.c_B.A'.m \ \mathbf{in}\ tr \Rightarrow \\
&\qquad \mathit{Feasible}_{tr}(\mathit{send}.A.c_A.B'.m, \mathit{receive}.B.c_B.A'.m) \vee \\
&\quad c_A \in \mathit{IntruderConnection} \wedge \forall m, tr' \cdot \\
&\qquad tr' {}^\frown \langle \mathit{receive}.B.c_B.A'.m \rangle \leqslant tr \Rightarrow \\
&\qquad\quad \mathit{fake}.A'.B.c_B.m \ \mathbf{in}\ tr' \,.
\end{aligned}$$

The connection c_B is related to every connection that could feasibly have sent (or faked) all of the messages that are received in c_B.

It is not hard to formulate valid system traces that induce non-functional relations, for example, if two agents send sequences of messages that share a common subsequence that is received by another agent.

For such traces there are different ways of interpreting the events and of resolving the non-determinism in the relation. Each of these interpretations is represented by a *maximal functional refinement* of $\mathcal{R}_{tr}$, i.e., any relation $\mathcal{R}'$ that: is a subset of $\mathcal{R}_{tr}$; has the same left-image as $\mathcal{R}_{tr}$; and is functional (where $\mathcal{R}_{tr}$ might not be). We write $\mathcal{R}' \ll \mathcal{R}_{tr}$ when $\mathcal{R}'$ is a maximal functional refinement of $\mathcal{R}_{tr}$. Note that $\ll$ is not necessarily reflexive, but if $\mathcal{R}_{tr} \ll \mathcal{R}_{tr}$ then $\mathcal{R}_{tr}$ is functional so has no proper refinements.

In order to specify session and stream properties we place conditions on the maximal functional refinements of the $\mathcal{R}$ relation. We will also restrict attention to a pair of roles: we define $\mathcal{R}_{tr} \upharpoonright (R_i \rightarrow R_j) = \mathcal{R}_{tr \upharpoonright \Sigma(R_i \rightarrow R_j)}$, and $\mathcal{R}_{tr} \upharpoonright \{R_i, R_j\} \mathrel{\widehat{=}} \mathcal{R}_{tr \upharpoonright \Sigma(R_i \rightarrow R_j) \cup \Sigma(R_j \rightarrow R_i)}$, where:

$$\begin{aligned}
&\Sigma(R_i \rightarrow R_j) = \{| \, \mathit{send}.\hat{R}_i.\mathit{Connection}.\hat{R}_j, \mathit{fake}.\hat{R}_i.\hat{R}_j, \\
&\quad \mathit{receive}.\hat{R}_j.\mathit{Connection}.\hat{R}_i, \mathit{hijack}.\hat{R}_i \rightarrow \hat{R}_i.\hat{R}_j \rightarrow \hat{R}_j \, |\} \,.
\end{aligned}$$

Session channels Consider the example implementation of a secure channel that satisfies $C \wedge \mathit{NR}^-$ given in Figure 2: $\{m\}_{PK(B)}, A$. There is nothing in the transport layer message to distinguish this message from one sent by *A* to *B* in a different connection. It is clear that if A does send two messages to B in different connections, the system will accept traces in which B receives them in a single connection. Further, since the intruder can fake messages on this channel, it is possible that B receives a mix of messages from A and from the intruder in the same session.

If A's protocol agent included a fresh nonce with the first message that *A* sent to *B*, and then sent that nonce with each subsequent message then as long as that nonce remains secret, neither of the attacks above are possible. The messages that *A* sends are bound together by the nonce, so B's protocol agent knows that each message it receives from A was sent in a single connection, and so will ensure that B receives them in a single connection. It is impossible for the intruder to inject messages into the connection as he does

not know the nonce. In order to fake messages the intruder must bind them together with a single nonce; in this case they can be thought of as coming from a single dishonest connection.

This modification allows us to use this transport layer protocol to establish sessions: all of the messages sent in a single connection will be received in a single connection, and the intruder cannot inject messages into the session. The intruder can still remove and re-order messages within the session.

Definition 5.1 (Session)**.** A channel $R_i \rightarrow R_j$ is a session channel if the relation $\mathcal{R}_{tr} \restriction (R_i \rightarrow R_j)$ is left-total:

$$\mathit{Session}(R_i \rightarrow R_j)(tr) \mathrel{\widehat{=}} \\ \forall c_B \cdot \exists c_A \cdot (c_B, c_A) \in \mathcal{R}_{tr} \restriction (R_i \rightarrow R_j)\,.$$

The transport layer protocol described above has the unfortunate property that the intruder can replay messages from old sessions, and cause B to believe that A wishes to start a new session with him. This is an attack against the injectivity of the transport protocol. Similar attacks are possible against other transport protocols in which, for example, one session is played to two different agents. The next property prevents this sort of attack.

Definition 5.2 (Injective Session)**.** A channel $R_i \rightarrow R_j$ is an injective session channel if it is a session channel and there exists an injective maximal functional refinement $\mathcal{R}'$ of the *receives-from* relation $\mathcal{R}_{tr} \restriction (R_i \rightarrow R_j)$:

$$\mathit{InjectiveSession}(R_i \rightarrow R_j)(tr) \mathrel{\widehat{=}} \\ \mathit{Session}(R_i \rightarrow R_j)(tr) \wedge \\ \exists \mathcal{R}' \ll \mathcal{R}_{tr} \restriction (R_i \rightarrow R_j) \cdot \mathcal{R}' \text{ is injective}\,.$$

The security parameters of a TLS connection are used to protect the integrity of every record layer message. This integrity check, and the secrecy of the security parameters ensures that TLS is a session channel. The agents calculate the security parameters together in the handshake, so they both know they have contributed to the values of the security parameters, and so they are communicating in a new session. In order to replay a TLS session the intruder would have to be able to choose the security parameters to match those of the old session. TLS therefore establishes injective sessions.

The TLS handshake protocol ensures that the connections held by the client and server are bound together in a single session. However, it is not the case that every injective session channel achieves this. It is possible for the intruder to interleave connections in such a way that a message A receives in a connection c_A may not be in response to the messages she sent in c_A.

Definition 5.3 (Strong Session)**.** A channel $R_i \rightarrow R_j$ is a strong session channel if the channels $R_i \rightarrow R_j$ and $R_j \rightarrow R_i$ are session channels and there exists a symmetric maximal functional refinement $\mathcal{R}'$ of the *receives-from* relation $\mathcal{R}_{tr} \restriction \{R_i, R_j\}$:

$$\mathit{StrongSession}(R_i \rightarrow R_j)(tr) \mathrel{\widehat{=}} \\ \mathit{Session}(R_i \rightarrow R_j)(tr) \wedge \mathit{Session}(R_j \rightarrow R_i)(tr) \wedge \\ \exists \mathcal{R}' \ll \mathcal{R}_{tr} \restriction \{R_i, R_j\} \cdot \forall c_A, c_B \cdot c_B \mathcal{R}' c_A \Rightarrow c_A \mathcal{R}' c_B\,.$$

The strong session property ensures that the sort of session interleaving and de-coupling described above cannot happen. A strong session channel also ensures injectivity.

Stream channels The record layer of TLS includes a further integrity protection mechanism: sequence numbers. The sequence number is authenticated by the usual TLS integrity protection, so the recipient of a TLS stream can be sure that he has not missed any messages, nor received messages in any order other than that intended by the sender.

TLS therefore provides a stronger guarantee than the strong session property: 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 permuting the order in which messages are received, or inserting or removing messages from a session. However, the intruder can terminate a stream at any point.

We define stream channels by altering the definition of the *receives-from* relation $\mathcal{R}_{tr}$ to form the *stream-receives-from* relation: $\mathcal{S}_{tr}$. $c_B \mathcal{S}_{tr} c_A$ if the stream of messages received in c_B is a prefix of the stream of messages sent in c_A. *Stream*, *injective stream* and *strong stream* channels are defined in the same way as the session channels using $\mathcal{S}_{tr}$ in place of $\mathcal{R}_{tr}$.

$$c_B \mathcal{S}_{tr} c_A \mathrel{\widehat{=}} \forall A', B \cdot \\ \exists A, B' \cdot tr \downarrow \mathit{receive}.B.c_B.A' \leqslant tr \downarrow \mathit{send}.A.c_A.B' \wedge \\ \forall m \cdot \mathit{receive}.B.c_B.A'.m \textbf{ in } tr \Rightarrow \\ \mathit{Feasible}_{tr}(\mathit{send}.A.c_A.B'.m, \mathit{receive}.B.c_B.A'.m) \vee \\ c_A \in \mathit{IntruderConnection} \wedge \\ tr \downarrow \mathit{receive}.B.c_B.A' \leqslant tr \downarrow \mathit{fake}.A'.B.c_B\,.$$

We believe that TLS (in unilateral and bilateral mode) establishes strong stream channels [7].

It is clear that $c_B \mathcal{S}_{tr} c_A \Rightarrow c_B \mathcal{R}_{tr} c_A$; hence each of the stream channels is simulated by the equivalent session channel. These six properties give rise to the hierarchy shown in Figure 3.

We believe that each of the channels in Figure 2 except the bottom one can be strengthened to give a session property by including a session identifier in the transport-layer message. However, this must be done with care; for example, the channel that sends messages as $\{\{m\}_{SK(A)}\}_{PK(B)}$ cannot be strengthened to a session channel by including a session identifier outside the sender's signature, $\{\{m\}_{SK(A)}, c_A\}_{PK(B)}$, as this would allow the intruder to take messages from two different sessions between A

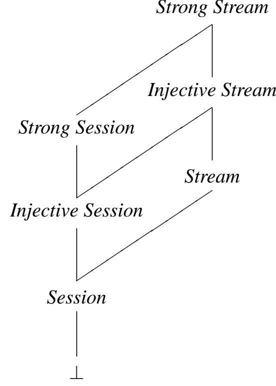


Figure 3. The session and stream channel hierarchy.

and himself, and combine them into a new session between A and some other honest agent. Instead, the session identifier must be bound to the application-layer message: $\{\{m, c_A\}_{SK(A)}\}_{PK(B)}$.

Some of the channels can be further strengthened to give a strong session property. We also believe that any session channel can be strengthened to give a stream property, and any injective or strong session channel can be strengthened to give an injective or strong stream property by binding authenticated sequence numbers to every message, as in TLS.

6. Simulation

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 this section we present a simulation relation that compares channel specifications by comparing the honest agents' views of them. We justify this definition, and use it to establish an equivalence relation (simulation in both directions) on channel specifications.

Process simulation is usually defined in terms of a simulation relation between the states of processes. However, we use the term simulation to mean something different: we want to capture the notion that one channel allows the same attacks as another. If specification P simulates specification Q, then P allows every attack that Q allows; in other words, P is no more secure than Q.

Subset inclusion of the traces of our channel specifications would capture too much information for the simulation relation discussed above. For example, a channel in which the intruder cannot perform the fake event, but can hijack and re-ascribe his own messages should certainly simulate a nearly identical channel in which the intruder can fake messages. However, there are traces that the latter specification accepts that the former does not (e.g. any trace that contains a fake event). In order to draw the correct conclusion about these two specifications we need to look at the result of the intruder's behaviour, and not the way in which he performs it. Rather than directly comparing the traces of two specifications we must compare the honest agents' views of the traces.

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:

$$\begin{aligned} &\mathit{HonestTraces}_{IIK}(\mathit{ChannelSpec}) \mathrel{\widehat{=}} \\ &\quad \{tr \upharpoonright \{| \mathit{send.Honest}, \mathit{receive.Honest} |\} \mid \\ &\qquad tr \in \mathit{traces}_{IIK}(\mathit{ChannelSpec})\}\,. \end{aligned}$$

Definition 6.1 (Simulation)**.** The channel specification $\mathit{ChannelSpec}_1$ simulates $\mathit{ChannelSpec}_2$ 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:

$$\begin{aligned} &\forall \mathit{IIK} \subseteq \mathit{Message} \cdot \mathit{HonestTraces}_{IIK}(\mathit{ChannelSpec}_2) \subseteq \\ &\quad \mathit{HonestTraces}_{IIK}(\mathit{ChannelSpec}_1)\,. \end{aligned}$$

We write $\mathit{ChannelSpec}_1 \preccurlyeq \mathit{ChannelSpec}_2$.

If $\mathit{Spec}_1 \preccurlyeq \mathit{Spec}_2$ we claim that the intruder can perform any attack on the first specification that he can on the second (i.e. the first specification is no more secure than the second). This is clearly true for those attacks that can be detected by looking at the honest traces. The result is not so clear for attacks that cannot immediately be detected by looking at the honest traces; in particular, in order to detect attacks against confidentiality we must examine the intruder's knowledge after traces of the specifications. We can show that if there is a fact f that the intruder can learn under a specification (either by performing a legitimate protocol run with another agent, or by learning a secret), then he should be able to learn that fact under any specification that simulates the first.

Proposition 6.2. *If* $\mathit{ChannelSpec}_1 \preccurlyeq \mathit{ChannelSpec}_2$ *then:*

$$\begin{aligned} &\forall \mathit{IIK}_2 \subseteq \mathit{IIK}_1 \subseteq \mathit{Message} \cdot \\ &\quad \forall tr_2 \in \mathit{traces}_{IIK_2}(\mathit{ChannelSpec}_2) \cdot \\ &\qquad \exists tr_1 \in \mathit{traces}_{IIK_1}(\mathit{ChannelSpec}_1) \cdot \\ &\qquad\quad \mathit{HonestTrace}(tr_2) = \mathit{HonestTrace}(tr_1) \wedge \\ &\qquad\quad \mathit{IntruderKnows}(tr_2) \subseteq \mathit{IntruderKnows}(tr_1)\,. \end{aligned}$$

We define our equivalence relation as simulation in both directions.

Definition 6.3 (Equivalence)**.** Two channel specifications $\mathit{ChannelSpec}_1$ and $\mathit{ChannelSpec}_2$ are equivalent if they simulate each other. We write $\mathit{ChannelSpec}_1 \cong \mathit{ChannelSpec}_2$.

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.

7. Alternative channel specifications

We have specified our channels by blocking the dishonest events that the intruder can perform. Specifying the channels in this way gives a simple set of definitions; once the intruder's initial powers are understood, it is easy to see the restrictions that are created by blocking, or limiting, his use of one of the dishonest events. However, the specifications are not particularly useful for proving properties about systems. In this section we give alternative formulations for our channel specifications; these alternatives state exactly which events must have occurred before an honest agent can receive a message, and are more conducive to proving properties about our networks.

Network rule ($\mathcal{N}_4$) states the necessary events that must have happened before an honest agent can receive a message: when an honest agent B receives message m, apparently from agent A, then either A really sent that message to B, the intruder faked the message, or the intruder has hijacked a message and caused B to receive it from A. None of our basic authentication channels prevent messages from being replayed, but the strongest channel ($\mathit{NF} \wedge \mathit{NRA} \wedge \mathit{NR}$) prevents all other hijack events, so this channel satisfies a stronger form of $\mathcal{N}_4$ in which the only possibility is that A really did send the message to B:

$$\begin{aligned}&\mathit{StrongAuth}(R_i \rightarrow R_j)(tr) \mathrel{\widehat{=}} \forall B : \hat{R}_i, c_B, A : \hat{R}_i, m \cdot \\ &\quad \mathit{Honest}(B) \wedge \mathit{receive}.B.c_B.A.m \ \mathbf{in}\ tr \Rightarrow \\ &\qquad \exists c_A \cdot \mathit{send}.A.c_A.B.m \ \mathbf{in}\ tr\,.\end{aligned}$$

In this case, it was obvious how to form the alternative specification for the channel: none of the dishonest events is allowed (except a replay), so none of them could have been the cause of the receive event. We note that this alternative form of the specification does not prevent the intruder from performing dishonest events on the channel (except replays); however, any dishonest event that he does perform cannot cause an honest agent to receive a message that they would not otherwise have received (as there must also be a send event).

The alternative specification for any combination of the channel primitives is formed as below:

- The alternative form of no-faking (*NF*) is formed by removing the $\mathit{fake}.A.c_A.B.m$ possibility from $\mathcal{N}_4$;
- The alternative form of no-re-ascribing (*NRA*) must not allow message re-ascribing: if the receive event was caused by $\mathit{hijack}.A' \rightarrow A.c_A.B' \rightarrow B.c_B.m$ then $A = A'$;
- The alternative form of no-honest-re-ascribing (NRA^-) must restrict the possibilities for message re-ascribing: if the receive event was caused by $\mathit{hijack}.A' \rightarrow A.B' \rightarrow B.c_B.m$ then $A = A'$ or A must be dishonest;
- The alternative form of no-redirecting (*NR*) must not allow message redirection: if the receive event was caused by $\mathit{hijack}.A' \rightarrow A.B' \rightarrow B.c_B.m$ then $B' = B$;
- The alternative form of no-honest-redirecting (NR^-) must restrict the possibilities for messages redirection: if the receive event was caused by $\mathit{hijack}.A' \rightarrow A.B' \rightarrow B.c_B.m$ then $B = B'$ or B' must be dishonest.

One can prove that each of the alternative forms is equivalent to the corresponding original specification. For example, let

$$\begin{aligned}&\mathit{SenderAuth}(R_i \rightarrow R_j)(tr) \mathrel{\widehat{=}} \forall B : \hat{R}_j, c_B, A : \hat{R}_i, m \cdot \\ &\quad \mathit{Honest}(B) \wedge \mathit{receive}.B.c_B.A.m \ \mathbf{in}\ tr \Rightarrow \\ &\qquad \exists c_A : \mathit{Connection} \cdot \mathit{send}.A.c_A.B.m \ \mathbf{in}\ tr \vee \\ &\qquad \exists A' : \hat{R}_i, B' : \hat{R}_j \cdot (\mathit{Dishonest}(A) \vee A = A') \wedge \\ &\qquad\quad \mathit{hijack}.A' \rightarrow A.B' \rightarrow B.c_B.m \ \mathbf{in}\ tr\,.\end{aligned}$$

Theorem 7.1. *The following specifications are equivalent:*

$$\begin{aligned}\mathit{ChannelSpec}_1 &= \mathit{ChannelSpec} \wedge \mathit{SenderAuth}(R_i \rightarrow R_j)\,, \\ \mathit{ChannelSpec}_2 &= \mathit{ChannelSpec} \wedge (\mathit{NF} \wedge \mathit{NRA}^-)(R_i \rightarrow R_j)\end{aligned}$$

where ChannelSpec is any channel specification.

8. 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, we see their main use as being to analyse layered security architectures. 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. [6]. We propose to analyse security protocols that use secure transport layers using Casper [8] and FDR [10]; to do so, we will build CSP models capturing the services provided by secure channels. 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 and transport-layer messages 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 also intend to investigate how secure channels can be combined together, either chained together in series or layered one on top of another: what properties are satisfied by such combinations?

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

Broadfoot and Lowe In [3], Broadfoot and Lowe specify a form of secrecy that is equivalent to our network with every channel being non-re-ascribable and confidential.

The difference between our definition of confidentiality and that given in [3] is that we allow the intruder to change the identity of the sender of a message. In the model in which Broadfoot and Lowe's results should be interpreted, the intruder does not possess this capability, so their definition ought to be compared to, and is equivalent to, non-re-ascribable confidential channels.

Broadfoot and Lowe also specify a single form of authenticated channel, which is equivalent to an authenticated stream channel.

Empirical channels Creese et al. have developed the notion of *empirical channels*, and adapted the traditional attacker model for analysing protocols in order to study security protocols for pervasive computing [4]. 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, such as two humans comparing a code printed on each of their laptop screens.

Over such channels, they specify any combination of the following restrictions on the intruder: no spoofing, no over-hearing and no blocking. No spoofing corresponds to our definition of an unfakeable channel, although Creese et al. allow two different models of this channel: one that allows redirecting, and one that doesn't. No overhearing corresponds to our definition of confidential channels. 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. With the empirical channels suggested in [4] it is easier to see how this would be possible.

Security architectures using formal methods Boyd [2] defines two different types of channel in a security architecture consisting of users and information about who trusts whom. A channel is a relationship between two users; a channel between two users is established when they share knowledge of a public key or a shared secret (e.g. a symmetric key). Thus channels are established by utilising existing keys, or propagating new keys between the two users wishing to communicate — often the propagation is over existing channels between trusted users. Boyd considers two types of cryptographic keys: *Confidentiality*, where only the intended user (or set of users) in possession of the secret key can read the message; and *Authentication*, where only that user (or set of users) in possession of the secret key can write the message.

Boyd's channels can either be symmetric (in which case each user is sure of the other's identity) or not (in which case one user may be unsure of the other's identity). On a non-symmetric confidentiality channel, the message receiver may be unaware of the sender's identity: this is equivalent to a confidential channel in our model. On a symmetric confidentiality channel the message sender is authenticated to the receiver: this corresponds to an authenticated confidential channel. A non-symmetric authentication channel is redirectable, while a symmetric authentication channel is not. Boyd's authentication is equivalent to a channel with either sender authentication or authenticaion.

Boyd's channels can be directly compared with some of our channels, but his reasons for specifying the channels are different to ours. Boyd specifies his channels to describe security architectures in terms of the secure channels available; the model describes when new channels may be established, and formalises some intuitively obvious results (for example that no secure channels can be established between users who possess no secrets); we specify channels in order to enrich our abstract layered model for protocol analysis.

Acknowledgements.

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

References

[1] M. Abadi. Two facets of authentication. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 25–32, 1998.

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

[3] 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.

[4] S. Creese, M. Goldsmith, R. Harrison, A. Roscoe, P. Whittaker, and I. Zakiuddin. Exploiting empirical engagement in authentication protocol design. *International Conference on Security in Pervasive Computing, Lecture Notes in Computer Science*, 3450:119–133, 2005.

[5] J. Guttman and F. Fàbrega. Protocol independence through disjoint encryption. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 24–34, 2000.

[6] S. Hansen, J. Skriver, and H. Nielson. Using static analysis to validate the SAML Single Sign-On Protocol. In *Proceedings of the Workshop on Issues in the Theory of Security*, 2005.

[7] A. Kamil and G. Lowe. Analysing TLS in the strand spaces model. *In preparation*, 2008.

[8] G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1–2):53–84, 1998.

[9] 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/.

[10] A. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

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

[12] 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.

[13] 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. Notation

An event is an atomic communication between processes; an event may carry data (for example $c.1$). A trace is a sequence of events that a 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 $\langle a.1, b.1, a.2, b.2, a.3 \rangle \downarrow a = \langle 1, 2, 3 \rangle$. $tr \upharpoonright c$ is the sequence of events communicated in tr restricted to the set of events c; for example, $\langle a.1, b.1, a.2, b.2, a.3 \rangle \upharpoonright \{| \ a \ |\} = \langle a.1, a.2, a.3 \rangle$.