On Distributed Security Transactions that use Secure Transport Protocols

Philippa Broadfoot Gavin Lowe*

February 7, 2003

Abstract

In this paper we consider techniques for designing and analysing distributed security transactions. We present a layered approach, with a high-level security transaction layer running on top of a lower-level secure transport protocol. The secure transport protocol provides protection against dishonest outsiders, while the transaction layer can be designed to provide protection against dishonest insiders. We specify generic services that one might expect such secure transport protocols to provide. We give examples of this layered approach, with the aim of demonstrating that the separation of concerns allows for a cleaner, more intuitive design. We consider how to analyse such a layered security architecture.

1 Introduction

In this paper we consider techniques for designing and analysing distributed security transactions. By a distributed security transaction we mean any transaction carried out between agents, over an insecure network such as the Internet, with some security-related goal; we will consider transactions with goals that go beyond simple authentication and key establishment. Example transactions include:

- An e-commerce protocol, that allows a customer to purchase goods from a merchant, with the cooperation of a bank (or payment gateway);
- A mechanism for resource access authorisation within a computational grid infrastructure, where distributed resources are shared within large, dynamic communities of users, called *virtual organisations* (VOs); access control

*Authors' address: Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom; e-mail {philippa.broadfoot, gavin.lowe}@comlab.ox.ac.uk.

policies are based upon membership of VOs or roles within VOs; the transaction therefore involves three parties: the resource itself, the resource acquirer, and a virtual organisation membership service (which provides credentials for resource acquirers, certifying their membership of the VO or roles within the VO);

- A mechanism for delegating the right to access certain resources, such as files or databases; this might be required in distributed computation, for example in a computational grid.

In such security transactions, there are two types of intruder or attacker that one needs to guard against:

Dishonest outsiders: dishonest agents who are not openly participating in this instance of the transaction, but who might instead imitate a participant; these are the intruders that one typically encounters in attacks upon authentication and key establishment protocols.

Dishonest insiders: dishonest agents who are openly participating in this instance of the transaction, using their own identities, but who attempt to subvert the transaction in some way; for example, a dishonest merchant in an e-commerce protocol might try to change the details of a transaction in some way, or to replay messages, so as to obtain more money than is due.

An obvious way to provide protection against a dishonest outsider is to protect all the communication channels using a secure transport protocol, such as SSL or TLS [Net96, SSL96, DA99, Tho00]. The security transaction itself can then be designed to only protect against dishonest insiders. This is the approach we consider in this paper. We aim to show that this way of considering protocols—separating issues of point-to-point authentication from those of the high level goals of the transaction—leads to clearer designs; we also consider the best way of analysing such designs.

Thus the architecture we consider is a layered one, with a high-level security transaction running on top of a lower-level secure transport protocol. This is depicted in Figure 1.

In the next section we consider what services the secure transport protocol should provide to the transaction layer. There is a trade-off here between what we would ideally like such a protocol to provide, and what it is reasonable to expect; our approach is to consider the services that appear to be provided by the protocols SSL and TLS [Net96, SSL96, DA99, Tho00].

In Section 3 we give some examples of secure transactions layered on top of a secure transport protocol; our aim is to demonstrate that the separation of concerns allows for a cleaner, more intuitive design.

In Section 4 we consider how to analyse such a security architecture. In principle, one could explicitly model the operation of the secure transport protocol

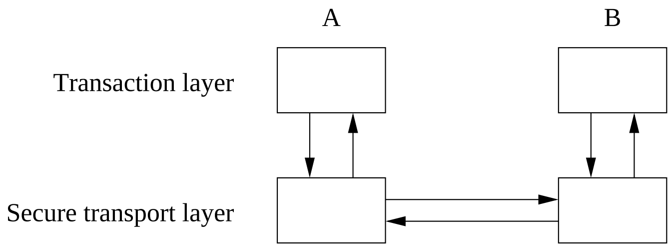


Figure 1: Layered protocols

within the Dolev-Yao model [DY83]. However, it is clearly more appropriate to abstract away from the details of the secure transport protocol, and model just the services it provides. This approach yields a conceptually simpler model than the Dolev-Yao model. It also makes automated analysis, for example using a model checker, more efficient: we briefly discuss our experiences in this area.

We conclude and discuss future work in Section 5.

2 What services should a secure transport protocol provide?

In this section we consider generic services that we believe should be provided by a secure transport layer protocol. Our approach can be used with *any* secure transport layer protocol that provides these services, although we will tend to motivate our discussions by reference to SSL and TLS. We do not distinguish between SSL and TLS in the rest of this paper, for they are nearly identical. We argue that SSL/TLS does indeed provide these services; proving this formally is the topic of current work.

The SSL/TLS protocol is typically configured to ensure one way authentication only (for example, when authenticating websites). We will, however, be considering the version that provides mutual authentication. The protocol uses the following two phases:

- An initial handshake, whereby both agents authenticate themselves, making use of public key certificates, and establish a fresh, secret session key; this part of the protocol is verified by Paulson in [Pau99];

- A record layer protocol, where the session key is used to provide secret, authenticated communication.

We consider the properties of the transport layer protocol with respect to the Dolev-Yao model [DY83]. In this model, the intruder can:

1. Overhear messages sent on the network;

2. Inject fake messages into the network, apparently from other agents;

3. Intercept messages passing on the network;

4. Take part in protocol runs using his own identity.

However, the intruder is assumed not to be able to perform cryptanalysis on the ciphertexts used.

Let's consider the above capabilities of an intruder when a secure transport protocol is used, to see how we would expect these capabilities to be restricted.

1. The intruder can overhear messages sent on the network; however, the secure transport protocol should ensure that the intruder cannot learn anything of import; for example, SSL/TLS encrypts all traffic using the secret session key.

2. The intruder should not be able to successfully inject messages into the network and have them accepted as coming from some third party; the secure transport protocol should be such that the recipient will detect the fake message; for example, SSL/TLS employs a message authentication code (MAC) for this purpose.

   Further, the intruder should not be able to replay messages, or reorder messages within a session; for example, SSL/TLS uses authenticated message numbers to prevent such attacks.

3. The intruder might be able to prevent messages reaching their intended destination, by intercepting the encrypted packet; further, the intruder might be able to delay messages from reaching their destination. It seems impossible for a protocol to protect against this, without physically securing all points on the transmission medium. However, it should not be possible for the intruder to delete messages from the middle of a session without the recipient detecting this; for example, the authenticated message numbers of SSL/TLS would allow the recipient to detect that messages had been missed.

4. The intruder can still take part in protocol runs using his own identity.

We seek to capture the services formally, using trace-based specifications: that is, we write a predicate with free variable tr representing a trace of communications; the intention is that the transport layer protocol ensures that *all* traces of the system satisfy this predicate.

Let $Agent$ be the set of all agents, partitioned into sets $Honest$ of honest agents, and $Dishonest$ of dishonest agents; more precisely, $Dishonest$ represents the set of identities that the intruder (or intruders) can adopt; if the transport protocol authenticates agents using public key certificates, then this will be the set of identities for which the intruder is in possession of a valid public key certificate and the corresponding private key.

We assume that the communications are grouped in sessions. The transport layer protocol should ensure that if A is in a session, apparently with B, then B really is in the same session, and vice versa. We assume some set $Session$ of sessions.

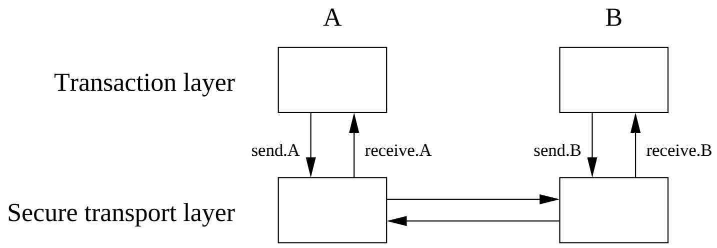


Figure 2: Layered protocols

Consider the layered architecture illustrated in Figure 2. We use two channels to describe the communications:

send The event $send.A.B.s.m$ represents the transaction layer at A passing to the corresponding secure transport layer the message m to be sent to B as part of session s.

receive The event $receive.B.A.s.m$ represents the transaction layer at B being passed by the secure transport layer the message m (apparently) received from A as part of session s.

Of course, the intruder can interfere with messages passing between the secure transport layer entities, but the transport layer protocol should be such that the *send* and *receive* messages satisfy suitable secrecy and authentication conditions; for example, the transport layer entities should detect and reject faked messages from the intruder.

We begin with the authentication and integrity requirements. One would expect that within each session between A and B, the messages accepted by the transport layer of B (and so passed to the transaction layer) as coming from A are a prefix of those sent by A intended for B[1]:

$$\forall A, B : Honest\,; s : Session \bullet tr \downarrow receive.B.A.s \leq tr \downarrow send.A.B.s\ . \qquad (1)$$

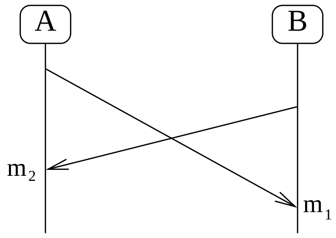
Note that this specification does not say that the agents are strongly synchronised: the sends and receives in different directions could be desynchronised,

[1] $tr \downarrow X$ represents the sequence of data items communicated on channel X; $\leq$ represents the prefix relation.

since the intruder can delay messages; for example, the specification allows the trace

$$\langle send.A.B.s.m_1, send.B.A.s.m_2, receive.A.B.s.m_2, receive.B.A.s.m_1\rangle,$$

illustrated below



Further, communications in different sessions might be interleaved in arbitrary ways. One might desire a stronger specification that forbids such traces, but we do not insist on that, for it is not a service that SSL/TLS provides.

We now consider the secrecy requirement, namely that no application data is leaked. We capture this requirement by specifying that all messages the intruder sends are messages that could be created by the intruder from the messages he has seen earlier and what he knew initially.

We define $sentToIntruder(tr)$ to be the set of messages deliberately sent to the intruder during trace tr:

$$\begin{array}{l} sentToIntruder(tr) \mathrel{\widehat{=}} \\ \quad \{m \mid \exists\, A : Agent\, ; I : Dishonest\, ; s : Session \bullet send.A.I.s.m \text{ in } tr\}. \end{array}$$

The intruder will be able to produce new messages from the set of messages that are sent to him. Following, for example, [RSG$^+$00], we write $Ms \vdash M$ if the intruder can deduce message M from the set of messages Ms. If we take the set of messages to be built up from atoms using concatenation and encryption, the relation $\vdash$ is defined by the following five rules:

member $M \in Ms \Rightarrow Ms \vdash M$,

concatenation $Ms \vdash M \wedge Ms \vdash M' \Rightarrow Ms \vdash (M, M')$,

splitting $Ms \vdash (M, M') \Rightarrow Ms \vdash M \wedge Ms \vdash M'$,

encrypt $Ms \vdash M \wedge Ms \vdash K \Rightarrow Ms \vdash \{M\}_K$,

decrypt $Ms \vdash \{M\}_K \wedge Ms \vdash K^{-1} \Rightarrow Ms \vdash M$.

Let IIK represent the initial knowledge of the intruder, i.e. those messages that the intruder can be assumed to know before taking part in protocol transactions.

We capture the secrecy requirement by specifying that an agent B can receive message m from the intruder only if that message can be produced from the intruder's initial knowledge and those messages that have previously been deliberately sent to him:

$$\begin{array}{l}\forall B : Honest\,;I : Dishonest\,;s : Session\,;m : Message\,;tr' : Trace \bullet \\ \quad tr' \frown \langle receive.B.I.s.m\rangle \leq tr \Rightarrow IIK \cup sentToIntruder(tr') \vdash m.\end{array} \tag{2}$$

For the rest of the paper, we will assume that the secure transport layer protocol satisfies the authentication and secrecy requirements captured by conditions (1) and (2).

3 Examples

In this section we consider two examples of security transactions that are layered on top of a secure transport protocol. We argue that, by assuming secure point-to-point channels, we are able to come up with simpler, clearer designs for such protocols.

3.1 A simple e-commerce protocol

We consider first a simple e-commerce protocol, to enable a customer C to purchase goods from some merchant M, over an insecure network, such as the Internet. The protocol will also involve a bank (or payment gateway) B, which we assume to be trustworthy. The main effect of a successful execution of the protocol will be that the bank transfers an appropriate amount of money from the customer's account to that of the merchant.

The security requirements are as follows:

Authentication Each agent needs to be authenticated to each other; they all need to agree upon the identities of all agents running the protocol, and the amount of money involved; the customer and merchant need to agree upon the details of the order; the customer and merchant both need to agree with the bank on the result of the transaction (whether it succeeded or not).

Secrecy The details of the transaction should not be leaked to a third party; further, we specify that the details of the order should be kept secret from the bank.

Non-repudiation The customer should be able to prove to a third party that the merchant agreed to the transaction, and the merchant should be unable to plausibly deny this; otherwise, a dishonest merchant could receive the payment and not deliver the goods. The protocol should be fair in the sense that the merchant should not be able to receive payment unless the customer receives evidence that can be used to show that the merchant agreed to the transaction.

> Further, the bank should be able to provide evidence that the customer agreed to the transaction.
>
> The merchant might need evidence that the customer received the goods, since a dishonest customer could deny this; this could, for example, be satisfied by the customer's physical signature upon delivery, but is out of scope of the protocol. Similarly, the mechanism by which the bank transfers the funds to the merchant is out of scope.

Below, we build on top of a secure transport layer to develop a protocol to achieve these requirements. We first develop a protocol to satisfy the authentication and secrecy requirements, and then extend it to achieve non-repudiation. The authentication and secrecy requirements are satisfied by the following very simple protocol, where each agent communicates directly with each other:

$$
\begin{array}{lcl}
\text{Message 1.} & C \rightarrow M & : B, \mathit{order}, \mathit{price}, \mathit{sessionId} \\
\text{Message 2.} & M \rightarrow B & : C, \mathit{price}, \mathit{sessionId} \\
\text{Message 3.} & C \rightarrow B & : M, \mathit{price}, \mathit{sessionId} \\
\text{Message 4.} & B \rightarrow M & : \mathit{result} \\
\text{Message 5.} & B \rightarrow C & : \mathit{result}\ .
\end{array}
$$

As a result of using a secure transport layer, all pairs of roles are mutually authenticated and the agents agree upon the values exchanged; further, the details are kept secret; no additional encryption, signatures or freshness guarantees are necessary.

Note that it is necessary to include the identity of the third party within each point-to-point connection (although not necessarily within every message of that connection), so as to prevent confusion between different concurrent sessions involving other third parties. The session identifier, *sessionId*, is to avoid confusion between different concurrent sessions involving the same three agents, so that the bank is able to tie together related connections.

Defining e-commerce protocols where all roles communicate directly with one another results in a very simple design, since most of the security is provided by the secure transport layer. However, this does not match well with the conventions of the standard sockets library. Instead we straighten out the protocol so that the customer and bank do not communicate directly, but instead communicate only via the merchant.

A straightened version of the above protocol would be:

$$
\begin{array}{lcl}
\text{Message 1.} & C \rightarrow M & : B, \mathit{order}, \mathit{price} \\
\text{Message 2.} & M \rightarrow B & : C, \mathit{price} \\
\text{Message 3.} & B \rightarrow M & : \mathit{result} \\
\text{Message 4.} & M \rightarrow C & : \mathit{result}\ .
\end{array}
$$

However, this does not provide adequate authentication between the customer and bank. Firstly, a dishonest merchant could simply send a fake message 2 to the bank, even if the corresponding customer were not participating in the protocol run. Secondly, a dishonest merchant could send an incorrect result back to the customer in message 4 (for example telling the customer that the transaction had failed, and tricking the customer into trying again).

This example highlights how the secure transport layer on its own does not protect the system from dishonest insiders: even though a dishonest insider is unable to fake a participant's identity, there is nothing stopping him from sending false information with his own identity.

To establish mutual authentication between the customer and bank, we arrange for them to sign messages. Furthermore, a guarantee of freshness is needed: even though the secure transport layer prevents a dishonest agent from replaying secure transport layer messages, there is nothing to stop a dishonest agent from replaying transaction layer messages from one run in another; for example, without a guarantee of freshness, a dishonest merchant could replay signed messages from one run in another, to obtain multiple payments. One solution is to use unique sequential numbers (analogous to cheque numbers), and to have the bank check that the sequence number has not been used before by this customer.

Taking these points into account, the protocol becomes:

$$\begin{array}{lrcl}
\text{Message 1.} & C \to M & : & \mathit{order}, \{B, \mathit{price}, \mathit{seqNr}\}_{SK(C)} \\
\text{Message 2.} & M \to B & : & \{B, \mathit{price}, \mathit{seqNr}\}_{SK(C)} \\
\text{Message 3.} & B \to M & : & \{C, \mathit{result}, \mathit{seqNr}\}_{SK(B)} \\
\text{Message 4.} & M \to C & : & \{C, \mathit{result}, \mathit{seqNr}\}_{SK(B)}\ .
\end{array}$$

Note that both the customer's identity and sequence number are necessary in messages 3 and 4, to prevent a dishonest merchant from replaying a result from a different run.

We now extend the protocol to consider non-repudiation properties. The non-repudiation evidence required by the bank—to prove that the customer agreed to the transaction—is already present in the protocol, in message 2. We add messages to the start of the protocol whereby the merchant gives the customer a signed commitment to the transaction before the customer commits. The merchant's commitment should be interpreted as a promise to fulfill the order *if* the correct payment is received; in the case of a dispute, the customer would call on the bank as a witness to verify that the payment had been made. The commitment includes a lifetime, limiting the validity of the promise.

$$\begin{array}{lrcl}
\text{Message 1.} & C \rightarrow M & : & \mathit{order},\ \mathit{price},\ \mathit{seqNr} \\
\text{Message 2.} & M \rightarrow C & : & \{\mathit{order},\ \mathit{price},\ \mathit{seqNr},\ \mathit{lifetime}\}_{SK(M)} \\
\text{Message 3.} & C \rightarrow M & : & \{B,\ \mathit{price},\ \mathit{seqNr}\}_{SK(C)} \\
\text{Message 4.} & M \rightarrow B & : & \{B,\ \mathit{price},\ \mathit{seqNr}\}_{SK(C)} \\
\text{Message 5.} & B \rightarrow M & : & \{C,\ \mathit{result},\ \mathit{seqNr}\}_{SK(B)} \\
\text{Message 6.} & M \rightarrow C & : & \{C,\ \mathit{result},\ \mathit{seqNr}\}_{SK(B)}\ .
\end{array}$$

3.2 Resource allocation protocols

We now consider two protocols, proposed by Foster *et al.* [FKTT98], for resource allocation in a computational grid architecture. The aim of a grid architecture (as described, for example, in [FK98]) is to enable the sharing of vast amounts of data and computational resources across highly distributed communities of users.

In order to use a grid, a user creates a *user proxy*, as a session manager process, and provides it with a signed credential together with associated secret key SK_{UP}; the credential is a short term certificate (for example in the form of an X.509 certificate), signed by the user, allowing the user proxy to authenticate itself, for example to establish an SSL/TLS session.

Both protocols below make use of a secure transport layer and a transaction layer; we show how these can be treated independently of one another using the framework discussed in this paper; this division also makes it easier to spot redundancy in the protocol.

Resource allocation protocol This protocol, Protocol 2 from [FKTT98], enables a user proxy *UP* to obtain a computational resource, and run a process upon it. Resource allocation requests are made to a *resource proxy RP*, rather than to the resource directly.

1. The user proxy *UP* and resource proxy *RP* authenticate each other using their credentials.
2. *UP* sends *RP* a signed allocation request.
3. *RP* checks whether the user who signed *UP*'s credential is authorised to make the allocation request under the local policy.
4. If successfully authorised, *RP* creates a public key PK_P for subsequent use by the process, and some information *ResourceInfo* about the resource being allocated; it returns these securely to *UP* using step 1.
5. *UP* verifies the information received in step 4 is as expected, and if so creates a signed credential of the form $\{\mathit{ResourceInfo}, PK_P\}_{SK_{UP}}$ for the resource, to allow it to subsequently re-authenticate itself to the user proxy.

6. UP passes the credential to RP securely, making use of step 1.

7. RP allocates the resource, and passes the credential to the newly created process; it is assumed that the resource proxy and resource are in the same trust domain, so this can be done securely.

The secure transport layer provides the mutual authentication between the user proxy and resource proxy, established in step 1, and used in subsequent steps. Furthermore, it provides the confidentiality necessary for the data transfer. If we abstract the secure transport layer away from the description above, we are left with the following transaction layer protocol:

$$
\begin{array}{lrcl}
\text{Step 2.} & UP \rightarrow RP & : & \{allocationRequest\}_{SK_{UP}} \\
\text{Step 4.} & RP \rightarrow UP & : & ResourceInfo, PK_P \\
\text{Step 6.} & UP \rightarrow RP & : & \{ResourceInfo, PK_P\}_{SK_{UP}} .
\end{array}
$$

This simpler view allows us to better understand the protocol. One interesting question that becomes apparent from this abstraction is the following: what is the purpose of the user proxy's signature in step 2? Foster *et al.* [FKTT98] do not answer this question, but given that communications are done over a secure transport layer, the signature is not necessary for authentication purposes; it is possible that it is included for purposes of non-repudiation.

The signature by the user proxy in step 6 is necessary, as it forms part of the credential that will be given to the new process. This credential will be used by the process for authentication purposes in any subsequent resource allocation requests.

Resource allocation from a process protocol We now consider a protocol that enables a process P acting for user proxy UP (as initialised by Protocol 2 above, for example) to obtain some other resource R' via its resource proxy RP'.

1. The process P and user proxy UP mutually authenticate using their credentials.

2. P sends UP a signed allocation request.

3. If UP decides to honour the request, then it runs Protocol 2 (described above) with the requested resource proxy RP'.

4. UP signs the resulting process handle and returns it to P.

Once again, the secure transport layer provides the necessary step-to-step authentication and secrecy. If we abstract the transport layer away from the description above, we are left with the following transaction layer protocol (we write, for example, “Step 3.4” for step 4 of Protocol 2 executed as step 3 of this protocol):

$$\begin{array}{lrcl}
\text{Step 2.} & P \to UP & : & \{allocationRequest_P\}_{SK_P} \\
\text{Step 3.2.} & UP \to RP' & : & \{allocationRequest_{UP}\}_{SK_{UP}} \\
\text{Step 3.4.} & RP' \to UP & : & ResourceInfo', PK_{P'} \\
\text{Step 3.6.} & UP \to RP' & : & \{ResourceInfo', PK_{P'}\}_{SK_{UP}} \\
\text{Step 4.} & UP \to P & : & \{ProcessHandle\}_{SK_{UP}} .
\end{array}$$

The same questions regarding signatures arise as in the previous case: the signatures on steps 2 and 4 appear unnecessary. Assuming the signature on step 3.2 is not necessary for non-repudiation, the protocol can be simplified to:

$$\begin{array}{lrcl}
\text{Step 2.} & P \to UP & : & allocationRequest_P \\
\text{Step 3.2.} & UP \to RP' & : & allocationRequest_{UP} \\
\text{Step 3.4.} & RP' \to UP & : & ResourceInfo', PK_{P'} \\
\text{Step 3.6.} & UP \to RP' & : & \{ResourceInfo', PK_{P'}\}_{SK_{UP}} \\
\text{Step 4.} & UP \to P & : & ProcessHandle .
\end{array}$$

4 Analysis

In this section we consider how best to model and analyse a secure transaction layered on top of a secure transport protocol. In principal one could explicitly model both the high and low level protocols, using the Dolev-Yao model. However, it is clearly better to abstract away from the details of the low level protocol, and consider just the services it provides to the high level protocol, as captured in Section 2.

We explain the analysis within the context of the CSP/FDR model [RSG$^+$00], for that is the analysis method we ourselves use; the basic idea is to build CSP models of the honest agents running the protocol, and of the most general intruder who can interact with them, and then to use the model checker FDR to explore the state space, looking for insecure states. In the conclusions, we briefly discuss prospects for adapting these ideas to other security protocol analysis techniques.

We begin by describing the standard CSP system model, i.e. when not using a secure transport protocol; we then explain how to adapt the model to capture the services provided by a secure transport protocol, and show that this adapted model meets the requirements from Section 2.

A brief introduction to CSP can be found in Appendix A.

4.1 The standard CSP model

The standard CSP model is depicted in Figure 3, where A and B are honest agents, and I is the intruder. For clarity, we include only two honest agents, and display only half the communications, namely those sent from A for B, from A to I, and from I to B, but in each case not those communications in the opposite direction. Note that all communications go through the intruder: he receives the results of all *send* events, and causes all *receive* events.

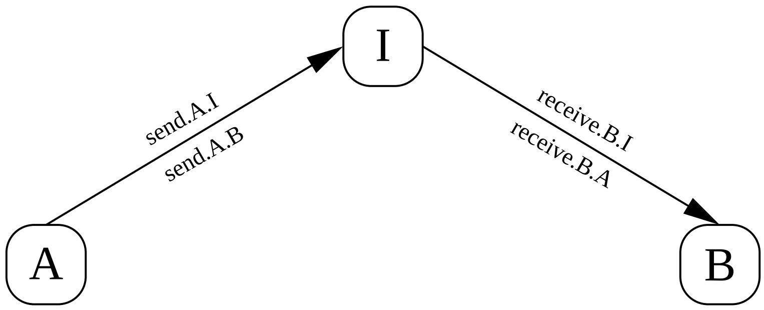


Figure 3: The traditional system model

The intruder process is such that he can send a message, causing that message to be received by an honest agent, only if it is deducible from the messages the intruder knew initially, and what he has observed on the network. If we define

$$
\begin{aligned}
&sentMessages(tr) \mathrel{\widehat{=}} \\
&\quad \{m \mid \exists\, A : Honest\,;\, B : Agent\,;\, s : Session \bullet send.A.B.s.m \text{ in } tr\}
\end{aligned}
$$

to be those messages sent during trace tr, then we can capture this property of the intruder as follows:

$$
\begin{aligned}
&\forall\, B : Honest\,;\, A : Agent\,;\, s : Session\,;\, m : Message\,;\, tr' : Trace \bullet \\
&\quad tr' \frown \langle receive.B.A.s.m \rangle \le tr \Rightarrow IIK \cup sentMessages(tr') \vdash m.
\end{aligned}
$$

See [RSG+00] for a definition of the intruder process.

4.2 The system model using a secure transport layer

By contrast, when the transaction is layered on top of a secure transport protocol, the appropriate model is as depicted in Figure 4.

The model of the intruder consists of two independent components: a component IK storing the intruder's knowledge, and performing runs of the protocol using the intruder's identity; and a component $BUFF_{A,B}$ delaying and possibly intercepting messages between A and B.

Note that:

- An event of the form $send.A.B.s.m$ (for A and B both honest) gets passed into the buffer $BUFF_{A,B}$; these messages might subsequently lead to corresponding $receive.B.A.s.m$ events, representing that the intruder can delay

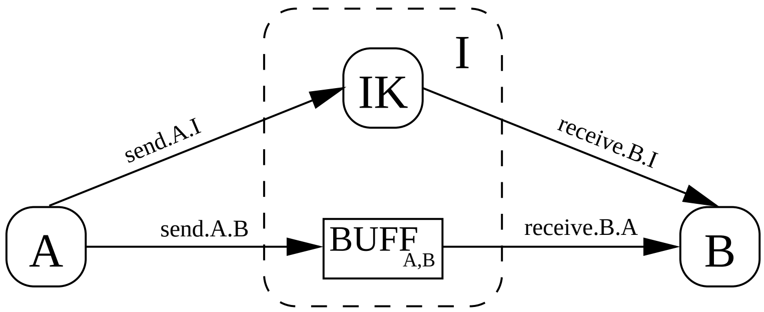


Figure 4: The new system model

messages; or the messages might never be delivered, representing that the intruder can intercept messages; we describe this buffer in more detail below.

- The intruder's knowledge, *IK*, does not synchronise on $send.A.B$ or $receive.B.A$ events, because these messages are encrypted, so he cannot learn anything from them.
- The intruder's knowledge can receive messages on channels $send.A.I$, for $A \in \mathit{Honest}$, but those are the only channels on which he can receive anything; this represents the fact that he cannot learn anything from overhearing other messages;
- *IK* can send messages on channels $receive.B.I$, for $B \in \mathit{Honest}$; this represents the fact that all communications are authenticated, so he can only send messages using his own identity.

The buffer between A and B acts as a FIFO queue for each session—corresponding to the fact that the intruder can delay messages between these agents—except at any point it can move into a state where no further messages are ever passed to B—corresponding to the fact that the intruder can prevent all messages reaching B from some point onwards. Note, in particular, that the buffer allows reordering between messages from different sessions, but no reordering within a session.

The buffer can be captured by the following CSP process; the component $BUFF_{A,B,s}(ms)$ captures the buffering for session s, where ms is the sequence of messages sent by A but not yet received by B.

$$
\begin{aligned}
& BUFF_{A,B} = \left|\left|\right|\right|_{s:Session} BUFF_{A,B,s}(\langle\rangle)\ , \\
& BUFF_{A,B,s}(ms) = \\
& \quad \left(\begin{array}{l} send.A.B.s?m \rightarrow BUFF_{A,B,s}(ms ^\frown \langle m \rangle) \\ \Box \\ ms \neq \langle\rangle \ \&\ receive.B.A.s.head(ms) \rightarrow BUFF_{A,B,s}(tail(ms)) \end{array} \right) \\
& \qquad \quad \rhd BUFF'_{A,B,s}\ , \\
& BUFF'_{A,B,s} = send.A.B.s?m \rightarrow BUFF'_{A,B,s}\ .
\end{aligned}
$$

(In fact, if we're interested only in safety properties, then we can omit the transfer to $BUFF'_{A,B,s}$, because it does not change the traces of the process.)

In principle, the two components of the intruder work together. However, both are nondeterministic processes, and so model checking will explore all possible ways of resolving that nondeterminism, which corresponds to all ways in which the two processes could be coordinated. Hence there is no need to explicitly model coordination between those processes.

4.3 Correctness of the model

We now show that the above model is correct, in the sense that its traces are precisely those traces that correspond to transaction layer entities communicating over channels that satisfy the authentication and secrecy conditions (1) and (2).

Firstly, note that[2]

$$
\begin{aligned}
traces(BUFF_{A,B,s}(\langle\rangle)) \ = \ & \{tr \in \{\!|receive.B.A.s, send.A.B.s|\!\}^* \mid \\
& \quad tr \upharpoonright receive.B.A.s \leq tr \upharpoonright send.A.B.s\}.
\end{aligned}
$$

Hence

$$
\begin{aligned}
traces(BUFF_{A,B}) \ = \ & \{tr \in \{\!|receive.B.A, send.A.B|\!\}^* \mid \\
& \quad \forall\, s : Session \bullet tr \upharpoonright receive.B.A.s \leq tr \upharpoonright send.A.B.s\}.
\end{aligned}
$$

Further, the intruder process is such that messages sent are derivable from messages received and those messages known initially; in the context of this system, this means:

$$
\begin{aligned}
& traces(INTRUDER) = \\
& \quad \{tr \in \{\!|receive.A.I, send.A.I \mid A \in Honest|\!\}^* \mid \\
& \qquad \forall\, B : Honest\,; I : Dishonest\,; s : Session\,; m : Message\,; tr' : Trace \bullet \\
& \qquad\quad tr' ^\frown \langle receive.B.I.s.m \rangle \leq tr \Rightarrow IIK \cup sentToIntruder(tr') \vdash m\}.
\end{aligned}
$$

[2] $tr \upharpoonright X$ represents the restriction of trace tr to those messages communicated on channel X. $\{\!|c|\!\}$ represents the set of all events on channel c.

Let the transaction layer entity of each honest agent A be modelled by a corresponding process P_A. Consider a parallel composition, as illustrated in Figure 4. We then immediately obtain, from the semantic definition of parallel composition (Appendix A), that the traces of the system are precisely those traces that satisfy:

$$
\begin{array}{l}
\forall A : \mathit{Honest} \bullet tr \upharpoonright \{\!|\mathit{send}.A, \mathit{receive}.A|\!\} \in \mathit{traces}(P_A) \\
\land \\
\forall A, B : \mathit{Honest}\,;\, s : \mathit{Session} \bullet tr \upharpoonright \mathit{receive}.B.A.s \leq tr \upharpoonright \mathit{send}.A.B.s \\
\land \\
\forall B : \mathit{Honest}\,;\, I : \mathit{Dishonest}\,;\, s : \mathit{Session}\,;\, m : \mathit{Message}\,;\, tr' : \mathit{Trace} \bullet \\
\qquad tr' \frown \langle \mathit{receive}.B.I.s.m \rangle \leq tr \Rightarrow \mathit{IIK} \cup \mathit{sentToIntruder}(tr') \vdash m.
\end{array}
$$

Note that the latter two conjuncts are conditions (1) and (2), respectively; hence this model precisely captures communication between transaction layer entities over channels satisfying these conditions.

4.4 On buffering

One might think that the buffering between the honest agents in Figure 4 is not always necessary. Many protocols are such that after any possible sequence of messages, there is at most one honest agent who could send the next message. For some time, we thought that for such protocols, one could omit the buffering, because allowing the buffering gave no advantage to the intruder. This turns out not to be the case.

Consider a protocol of the following form:

$$
\begin{array}{lrcl}
\text{Message 1.} & C \rightarrow B & : & m_1 \\
\text{Message 2.} & B \rightarrow C & : & m_2 \\
\text{Message 3.} & C \rightarrow A & : & m_3 \\
\text{Message 4.} & A \rightarrow B & : & m_4 \\
\text{Message 5.} & A \rightarrow C & : & m_5 \;.
\end{array}
$$

Suppose the intruder I takes the role of C in the protocol. Then the following exchange (where each M_i is an instantiation of the corresponding m_i) is possible if there is buffering between A and B, but would not be possible without buffering:

$$
\begin{array}{lrcl}
\text{Message 3.} & I \rightarrow A & : & M_3 \\
\text{Message 4.} & A \rightarrow I_B & : & M_4 \\
\text{Message 5.} & A \rightarrow I & : & M_5 \\
\text{Message 1.} & I \rightarrow B & : & M_1 \\
\text{Message 2.} & B \rightarrow I & : & M_2 \\
\text{Message 4}'. & I_A \rightarrow B & : & M_4 \;.
\end{array}
$$

I intercepts M_4 in message 4, and then sends it on in message $4'$. Note that each honest agent's view of this exchange does follow the protocol. This order of messages might give the intruder an advantage: by delaying M_4, I gets to learn M_5 before he has to produce M_1; having this additional knowledge might allow him to produce more values for M_5 than he would have been able to produce otherwise, without the buffering.

Another question involving buffering is the following: how much buffering do we need between A and B? If we are to perform model checking, then we can only model a finite capacity buffering if we are to keep the state space finite. Fortunately this is adequate, at least for bounded-length protocols.

Suppose the protocol description uses a message m from B to A; then this message will occur only after B has received all previous messages from A, and A will not send any more messages before receiving m; the effect of this is that B will send message m only in a state where the buffer from A to B is empty. Hence the maximum number of messages that can accumulate in the buffer is the maximum number of message in the protocol description from A to B without an intervening message from B to A.

Suppose the protocol description has n messages, numbered 1 to n, and write $k : A \to B$ if message k is sent from A to B. We are interested in subsequences $i \mathinner{..} j$ such that $1 \le i \le j \le n$ and such that the subsequence contains no messages from B to A, i.e. $\forall\, l \in i \mathinner{..} j \bullet \neg(l : B \to A)$. For such subsequences, all messages from A to B can accumulate in the buffer. The maximum number of messages that can accumulate in the buffer is therefore given by the following expression:

$$\max \{\, \#\{k \in i \mathinner{..} j \mid k : A \to B\} \;\mid\; 1 \le i \le j \le n \wedge \forall\, l \in i \mathinner{..} j \bullet \neg(l : B \to A) \,\}.$$

This is the required capacity for the buffer between A and B.

4.5 Experience

Casper [Low98] is a program that produces CSP models of protocols from more concise, abstract descriptions. We have extended Casper to support secret and authenticated channels, following the above description. We have also included support for channels that are just secret, or just authenticated.

We have analysed the example protocols described in Section 3 using Casper and FDR; we have not found any attacks, beyond those described earlier.

5 Conclusions

In this paper we have presented a methodology for designing and analysing secure transactions by layering them on top of a secure transport protocol. We have considered generic services that we would expect a secure transport protocol to provide, and argued informally that SSL/TLS does indeed provide these services. We have presented examples to illustrate that this separation of concerns provides a cleaner, simpler design for protocols. We have considered how to model and analyse such protocols within the CSP/FDR approach. In this final section we consider scope for extensions to this work.

As noted earlier, we intend to formally prove that SSL/TLS does indeed satisfy the requirements identified in Section 2.

The secure transport layer protocols we considered in this paper provided a weak form of synchronisation between directly communicating agents: in each direction, the messages received are a prefix of the messages sent. It would be interesting to consider a stronger transport protocol, that provides for stronger synchronisation between the agents, simultaneously synchronising messages in both directions, and providing an authenticated acknowledgement that each message has been received. We believe that this would make the problem of designing secure transactions even simpler: each acknowledged message would automatically provide agreement between the agents on those values contained in that message. Further it would provide for a simpler analysis technique: we believe that one would no longer need to provide buffering between the honest agents.

At would be interesting to adapt the ideas of this paper to other protocol analysis techniques. It should be clear how to adapt other model checking techniques, such as the NRL Protocol Analyzer [Mea96] or Murϕ [MMS97], by rearranging the channels as described in Section 4. For other techniques, such as those based upon theorem proving [Pau98, Coh00], Strand Spaces [FHG98] and Rank Functions [Sch97], it appears harder: it is easy enough to specify that the intruder receives only messages that are intended for it, and that honest agents accept only messages sent by the claimed sender; however, capturing the buffering between honest agents in a straightforward way seems harder, and likely to remove much of the elegance of those techniques.

In our analyses of specific protocols, mentioned in Section 4.5, we considered only small finite instances, i.e. with a fixed, finite number of honest agents. However, there is no reason why the layering approach cannot be extended with the data independence techniques of [RB99] to enable a more complete analysis. Further, we believe that there is scope for developing a completeness result, in the style of [Low99], where it is proved that under certain circumstances it is enough to analyse certain finite instances in order to deduce the security of all instances: the greater regularity provided by the use of a secure transport protocol seems likely to help us here.

We would also like to develop more experience of designing protocols using this methodology. To this end, and as part of our work on the EU-DataGrid, we intend to model and analyse some of the existing grid protocols, and design alternatives. An interesting question is whether the prudent design principles of Abadi and Needham [AN96] still hold, or whether new design principles are necessary.

One limitation of our approach is that we cannot analyse protocols that cut across the transaction and transport layers. For example, the first protocol from Section 3.2 established a credential, which was used to establish a secure transport connection in the second protocol. We can consider these protocols separately, but we cannot directly analyse them together within our framework.

Acknowledgements

We would like to thank Mike Auty and Nick Moffat for their helpful comments on this work. This work is partially funded by the UK Engineering and Physical Sciences Research Council as part of the e-science program.

References

[AN96] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[Coh00] Ernie Cohen. Taps: A first-order verifier for cryptographic protocols. In *Proceedings of 13th IEEE Computer Security Foundations Worksho p*, pages 144–158, 2000.

[DA99] T. Dierks and C. Allen. The TLS Protocol version 1.0. The Internet Engineering Task Force, 1999. RFC 2246.

[DY83] D. Dolev and A.C. Yao. On the security of public-key protocols. *Communications of the ACM*, 29(8):198–208, August 1983.

[FHG98] F.J.T. Fábrega, J.C. Herzog, and J.D. Guttman. Strand Spaces: Why is a security protocol correct? In *Proceedings of IEEE Symposium on Security and Privacy*, pages 160–171, 1998.

[FK98] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a new computing infrastructure.* Morgan Kaufmann, 1998.

[FKTT98] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *5th ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[Low98] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.

[Low99] Gavin Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2, 3):89–146, 1999.

[Mea96] C. Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[MMS97] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murϕ. In *IEEE Symposium on Security and Privacy*, 1997.

[Net96] Netscape Communications Corporation. *The SSL Protocol Version 3.0*, 1996.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[Pau99] Larry Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Computer and System Security*, 2(3):332–351, 1999.

[RB99] A.W. Roscoe and P.J. Broadfoot. Proving security protocols with model checkers by data indepen dence techniques. *Journal of Computer Security: Special Issue CSFW11*, 7(2,3):147–190, 1999.

[RSG+00] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols*. Pearson Education, 2000.

[Sch97] Steve Schneider. Verifying authentication protocols with CSP. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, 1997.

[SSL96] The SSL protocol. Available via `http://home.netscape.com/newsref/std/SSL.html`, 1996.

[Tho00] Stephen Thomas. *SSL and TLS Essentials*. Wiley, 2000.

A A brief overview of CSP

An event represents an atomic communication; this might either be between two processes or between a process and the environment. Channels carry sets of events; for example, $c.5$ is an event of channel c.

A *trace* of a process P is a sequence of events performed by P; $traces(P)$ is the set of all finite traces of P.

The process $a \rightarrow P$ can perform the event a, and then act like P. The process $c?x \rightarrow P_x$ inputs a value x from channel c and then acts like P_x. We write guarded processes in the form $b \mathbin{\&} P$; the process P is allowed to perform events only if b is true.

The process $P \,\square\, Q$ represents an external choice between P and Q; the initial events of both processes are offered to the environment; when an event is performed, that resolves the choice. The process $P \triangleright Q$ represents a transfer from P to Q: the initial events of P are offered for a period of time before control is transferred to Q.

The process $P \mid\mid\mid Q$ represents the interleaving of processes P and Q: the two processes run in parallel without synchronisation. This operator can also be distributed over a set: $\mid\mid\mid_{s:S} P_s$ represents replicated interleaving, indexed over the set S.

The process $P \;{}_A\|_B\; Q$ represents the alphabetised parallel composition of processes P and Q, with alphabets A and B respectively: P and Q are allowed to communicate events in their respective alphabets and they must synchronise upon events in the intersection $A \cap B$. This parallel operator has the following semantic definition (used in Section 4.3):

$$traces(P \;{}_A\|_B\; Q) = \{tr \in (A \cup B)^* \mid tr \upharpoonright A \in traces(P) \wedge tr \upharpoonright B \in traces(Q)\}.$$