Security for a Connectionless Peer-to-Peer Link

Graeme J. Proudler, Iris Harvey*
Mobile Communications Department
HP Laboratories Bristol
HPL-96-90
June, 1996

security, cellular radio, Internet, connectionless, client-server

This document describes a protocol designed to secure a connectionless communication channel between a mobile computer and a server. Specifically, the protocol checks that all messages have been delivered in the correct order to the correct destination, and a received message is the message that was sent. The protocol provides automatic authentication, integrity, and confidentiality on a communication link between two peers, specifically a mobile computer and a server. The protocol is simplified by the fact that key distribution is almost a null process because only two entities are involved, and those entities can at times communicate in a secure environment. A prototype is being built using a laptop PC with mobile phone as the mobile and a PC with modem as the server.

Internal Accession Date Only
**University of the West of England*, Coldharbour Lane, Frenchay, Bristol, United Kingdom

Security for a connectionless peer-to-peer link

Graeme Proudler, Hewlett Packard Laboratories Bristol,
Filton Road, Stoke Gifford, Bristol BS12 6QZ, UK.
Direct line: +44 117 9228 753, Switchboard: +44 117 9799 910
Fax: +44 117 9228 924, email: gjp@hplb.hpl.hp.com

Iris Harvey, University of the West of England,
Coldharbour Lane, Frenchay, Bristol, UK.

Abstract

This document describes a protocol designed to secure a connectionless communication channel between a mobile computer and a server[1]. Specifically,

- the protocol *checks* that all messages have been delivered in the correct order to the correct destination, and a received message is the message that was sent.

The protocol provides automatic authentication, integrity, and confidentiality on a communication link between two peers, specifically a mobile computer and a server. The protocol is simplified by the fact that key distribution is almost a null process because only two entities are involved, and those entities can at times communicate in a secure environment. A prototype is being built using a laptop PC with mobile phone as the mobile and a PC with modem as the server.

1 Introduction

The desirability of communicating-portable-computers depends on the services available and the cost of the communications. Howèver, any shortcomings in security are a barrier to popular use, and certainly restrict the applications that a prudent user will use.
The scenario is of a mobile user calling a server using a communication path that includes a cellular radio link and the PSTN or the internet. The path is a real-time duplex connection, albeit with latency. The conversation between the mobile and the server consists of a series of exchanges of messages. History teaches us that the average user is not willing to pay high prices for security, and it can be argued that the only commercially viable security system is one that provides personal security for an individual who has decided that he must have security:

- A user or application developer using existing (insecure) communication systems is unlikely to want to modify an existing communication package.

[1]The protocol is (naturally) also applicable to connection-oriented channels.

Such a person will be more comfortable adding a security layer immediately below his application, as this minimises the extra knowledge required. In effect, this option forces the user who requires a secure application to incorporate security into the application. The cost of security is then bourne by the user, rather than by the rest of the network.

Happily, it turns out that a secure peer-to-peer non-time-critical link is one of the simplest security situations. If the sole vunerability is the communication link, there is no need to secure individual machines. If the connection is always between a particular portable and a particular fixed machine, there is no difficulty in distributing security keys. And if data is non-time-critical and rates are relatively low, it is possible to run security processes on a standard processor. This security process sits immediately below an application and above the communication stack: the application sees it as the top of the communication stack and the communication stack sees it as the bottom of the application. Since the security process has become part of the communication stack, it follows that the security layer includes an auxillary function: it must retain a copy of a message until delivery of that message has been confirmed, in case retransmission is required.

2 Peer-to-peer security

This protocol uses many standard security techniques and mechanisms which are not explained here, and the interested reader is referred to [1] as an example of a ITU standard that uses authentication, and [2] for a comprehensive introduction to security in general.
It can be argued that the important security services required by the user are confidentiality, authentication, and integrity.

- *Confidentiality* renders the data unintelligable to an unauthorised person. It requires the use of algorithms such as DES etc. and careful choice is required in order to meet export controls.

- *Authentication* provides evidence of the identity of the peer at the other end of the communication path. Challenge/response mechanisms are often used to prove that the peer is 'on-line'.

- *Integrity* proves that the data has not been altered enroute. Integrity of individual messages can be established using checksums based on one-way 'hash' function (which have the advantage of having no export restrictions). Integrity of streams of messages can be determined by sequentially numbering messages and checking that they are delivered in order. This checks that messages are not repeated and that none are missing.

All of these services (and others) generally require the use of encryption and hence a mechanism to distribute encryption keys. This is in general an expensive and complicated process. But in this case the secure link is between just two points, which are both under the control of the user and can be in physical proximity when keys are exchanged, and key distribution is simple.
The protocol uses a hierarchy of encryption keys, to minimise the number of times that a human must establish 'trust' between two machines by loading the same key into both. In this case, the *password* is used to securely communicate *session keys* which securely communicate user data. This is the minimum number of keys that are required, and security could be increased if more keys were available (for instance, different session keys could be used for transmit and receive).
The server keeps a record of its secure communications. This is called the *audit* process, and is vital for security management.
Individual messages are built using encryption to provide confidentiality, a hash of an entire message to provide evidence of message integrity, and encryption of that hash to provide message authentication. The protocol (described next) is designed to provide stream integrity, and deals with messages which are repeated, missing or out-of-order.

3 The protocol

The protocol can simultaneously operate in both directions, so it does not matter which peer initiates a conversation, or even if both peers begin talking at the same time. It is built around two distinct states: the challenge state and the data state. A machine enters the challenge state when it is necessary to prove that a peer is on-line, and not a recording. User data is not sent during this time. A machine enters the data state when it is satisfied that messages are from a peer which is on-line. A machine therefore switches between these states.
The protocol ensures delivery in the correct order by use of specific acknowledgement messages or data-bearing messages from the peer that indicate reception of a message, and whether it was accepted or not. If the peer receives a message out-of-order, the security service will receive an acknowledgement to a later message before the acknowledgement to an earlier message. The security service therefore repeats the missing message and all messages after the missing message. It can safely do this because the peer has a mechanism for detecting out-of-order messages, and will ignore them (apart from sending an acknowledgement in reply)[2].
The ability to detect out-of-order messages is a natural consequence of the necessity to provide protection against replay attacks. The method is basically

[2]Note that if an acknowledgement was lost, and not the message, the only undesirable effect is the unnecessary retransmission of data.

just keeping track of a sequence number: a peer keeps a count of the sequence number of the last correctly received data-bearing message, and checks that received messages are processed in strict order.

3.1 Protocol details

The protocol requires a random number for initialising encryption of each message, a random number for the challenge process, and a random number as part of the sequence number. It uses the same random number for all three purposes.

- *Initilisation Vectors* are random numbers which are combined with a key in the encryption algorithm, so that changing the Initilisation Vectors causes the use of different sequences of the keystream even though the key may not have changed. IVs are sent with every message.

- Challenge processes 'prove' that a peer is on-line (ie. received messages are not old genuine messages). A unit invents a random number and sends it to the supposed peer. Then, if an incoming message passes authentication and integrity tests *and* contains that random number, the message must have come from a unit which is on-line *and* has the encryption keys which form the basis of the trust between the peers. So the message is accepted as coming from the peer unit.

- Sequencing schemes that count messages have difficulties when the counter wraps-around or it becomes necessary to reinitialise the counter. One solution is to prefix each sequence number with a random number, so the sequence number counts the number of times that a particular prefix has been used in the transmission of a data-bearing message.

  To achieve this, the sequence number is preset everytime that the prefix is changed, and incremented everytime after a new data-bearing message is sent. When the sequence number is about to roll over to zero, the prefix is always replaced and the sequence number is preset.

  However, when the prefix changes, a receiver must check that this has been done by a valid source. This is done by challenging the source. Only one challenge is necessary to check that a new prefix came from a valid source — afterwards the sequence number is sufficient to detect replays.

  Further, since a source needs to know which of its messages have been delivered, the prefix and sequence number of the last message received at a unit must be echoed back to the source in data-bearing messages or specific acknowledgement messages.

This particular protocol uses the same random numbers as prefixes, challenges, and initialisation vectors in an attempt to minimise the number of fields in a transmitted message. The protocol causes the random numbers to always change in pairs, and each message contains a pair of numbers which are simultaneously the transmit prefixes for both directions of a link, the challenges for both directions of the link, and the initialisation vectors for both directions of the link. Each node therefore determines a random number in every message on the link, no matter who transmitted that message. This allows each node to determine whether a received message is part of the current sequence of messages.
When a prefix is changed, messages with that prefix will be ignored by the receiver until the prefix has been validated, and sending ordinary data messages is pointless until that is done. This particular scheme therefore does not attempt to send normal data when a prefix is changed, and enters a separate protocol to ensure that both ends of a link are satisfied with the new prefix. The challenge protocol involves associating a pair of prefixes: one from received data and the other from transmitted data. When a machine changes its transmit prefix or receives a changed prefix it must build another pair, such that neither of the new prefixes appears in the old pair.
A forced exchange of 'challenge reply' messages is necessary because a transmitter has to find out the last *correctly received message* at the destination. The transmitter needs this information because it must retransmit any old messages (which used the old prefix) which have not been correctly received at the destination. However, any retransmitted messages must be sent using the new prefix and new sequence numbers (otherwise they will be rejected), which means that any duplicates of old messages resent in error will not be detected. The 'challenge reply' message is the first message containing an echo of a new prefix , and hence is the first message that is trusted by a receiver. Subsequent data bearing messages will also be trusted, but cannot indicate that the previous *correctly received message* used an old prefix (the message would be rejected). Hence the 'challenge reply' message is used to indicate the last *correctly received message* received with the previous prefix, by sending the old prefix and sequence number in the 'data' field of the 'challenge reply' message.[3] If a machine changes its transmit prefix, it is therefore necessary for a machine to repeatedly send 'challenge' messages until it receives an acceptable 'challenge response' message. The challenge protocol to achieve this has the following rules:

1. if a machine has changed its transmit prefix *for whatever reason*, repeatedly send challenge messages[4] containing

[3]If there is no 'last *correctly received message*' with the previous prefix, the 'challenge reply' sends the new prefix and a zero sequence number in its 'public data' field.

[4]T_α is the new prefix of the source of this message, R_β is the prefix last received at the source of this message.

$$T_\alpha = \text{new},\ R_\beta = 0$$

until a challenge reply is received.

2. for every challenge received, send a single challenge reply. A valid challenge reply is defined by

$$T_\beta = \text{new},\ R_\alpha = \text{new},$$

   - if the T prefix received in the challenge is an unassociated prefix:
     - if an unassociated local prefix already exists, associate the T prefix with that local prefix and send a challenge reply using those prefixes.
     - if an unassociated local prefix does not already exist, create a new unassociated local prefix, associate the T prefix with that local prefix and send a challenge reply using those prefixes. Enter the challenge state and send 'challenge' messages using the new local prefix until a 'challenge reply' is received.
   - if the T prefix received in the challenge is an associated prefix, send a challenge reply using those associated prefixes.

The system can now return to transmission of normal data. The sequence number is one in the first data-bearing message after a change of prefix, and thereafter incremented everytime that a new data-bearing message is sent.
If both peers maintain these prefixes and sequence numbers between connections, the challenge/response mechanism will operate only when a sequence number rolls over. If a machine chooses to change the random number in its sequence ID (perhaps because it does not remember state between connections), the challenge/response mechanism will operate when a new connection is initiated. Thus a machine does not have to remember this extra state information, but pays a price in the form of extra messages for the challenge/response protocol.

References

[1] Recommendation X.509 *'The Directory - Authentication Framework'*, ITU.

[2] Bruce Schneier *'Applied Cryptography'*, second edition (1996), Wiley.