

## A Distributed-Protocol Authentication Scheme

### Status of this Memo

The purpose of this RFC is to focus discussion on authentication problems in the Internet and possible methods of solution. The proposed solutions in this document are not intended as standards for the Internet at this time. Rather, it is hoped that a general consensus will emerge as to the appropriate solution to authentication problems, leading eventually to the adoption of standards. Distribution of this memo is unlimited.

### 1. Introduction and Overview

This document suggests mediated access-control and authentication procedures suitable for those cases when an association is to be set up between multiple users belonging to different trust environments, but running distributed protocols like the existing Exterior Gateway Protocol (EGP) [2], proposed Dissimilar Gateway Protocol (DGP) [3] and similar protocols. The proposed procedures are evolved from those described by Needham and Shroeder [5], but specialized to the distributed, multiple-user model typical of these protocols.

The trust model and threat environment are identical to that used by Kent and others [1]. An association is defined as the end-to-end network path between two users, where the users themselves are secured, but the path between them is not. The network may drop, duplicate or deliver messages with errors. In addition, it is possible that a hostile user (host or gateway) might intercept, modify and retransmit messages. An association is similar to the traditional connection, but without the usual connection requirements for error-free delivery. The users of the association are sometimes called associates.

The proposed procedures require each association to be assigned a random session key, which is provided by an authentication server called the Cookie Jar. The procedures are designed to permit only those associations sanctioned by the Cookie Jar while operating over arbitrary network topologies, including non-secured networks and broadcast-media networks, and in the presence of hostile attackers. However, it is not the intent of these procedures to hide the data

(except for private keys) transmitted via these networks, but only to authenticate messages to avoid spoofing and replay attacks.

The procedures are intended for distributed systems where each user  $i$  runs a common protocol automaton using private state variables for each of possibly several associations simultaneously, one for each user  $j$ . An association is initiated by interrogating the Cookie Jar for a one-time key  $K(i,j)$ , which is used to encrypt the checksum which authenticates messages exchanged between the users. The initiator then communicates the key to its associate as part of a connection establishment procedure such as described in [3].

The information being exchanged in this protocol model is largely intended to converge a distributed data base to specified (as far as practical) contents, and does not ordinarily require a reliable distribution of event occurrences, other than to speed the convergence process. Thus, the model is intrinsically resistant to message loss or duplication. Where important, sequence numbers are used to reduce the impact of message reordering. The model assumes that associations between peers, once having been sanctioned, are maintained indefinitely. The exception when an association is broken may be due to a crash, loss of connectivity or administrative action such as reconfiguration or rekeying. Finally, the rate of information exchange is specifically designed to be much less than the nominal capabilities of the network, in order to keep overheads low.

## 2. Procedures

Each user  $i$  is assigned a public address  $A(i)$  and private key  $K(i)$  by an out-of-band procedure beyond the scope of this discussion. The address can take many forms: an autonomous system identifier [2], an Internet address [6] or simply an arbitrary name. However, no matter what form it takes, every message is presumed to carry both the sender and receiver addresses in its header. Each address and its access-control list is presumed available in a public directory accessible to all users, but the private key is known only to the user and Cookie Jar and is not disclosed in messages exchanged between users or between users and the Cookie Jar.

An association between  $i$  and  $j$  is identified by the bitstring consisting of the catenation of the addresses  $A(i)$  and  $A(j)$ , together with a one-time key  $K(i,j)$ , in the form  $[A(i),A(j),K(i,j)]$ . Note that the reciprocal association  $[A(j),A(i),K(j,i)]$  is distinguished only by which associate calls the Cookie Jar first. It is the intent in the protocol model that all state variables and keys relevant to a previous association be erased when a new association is initiated and no caching (as suggested in [5]) is allowed.

The one-time key  $K(i,j)$  is generated by the Cookie Jar upon receipt of a request from user  $i$  to associate with user  $j$ . The Cookie Jar has access to a private table of entries in the form  $[A(i),K(i)]$ , where  $i$  ranges over the set of sanctioned users. The public directory includes for each  $A(i)$  a list  $L(i) = \{j_1, j_2, \dots\}$  of permitted associates for  $i$ , which can be updated only by the Cookie Jar. The Cookie Jar first checks that the requested user  $j$  is in  $L(i)$ , then rolls a random number for  $K(i,j)$  and returns this to the requestor, which saves it and passes it along to its associate during the connection establishment procedure.

In the diagrams that follow all fields not specifically mentioned are unencrypted. While the natural implementation would include the address fields of the message header in the checksum, this raises significant difficulties, since they may be necessary to determine the route through the network itself. As will be evident below, even if a perpetrator could successfully tamper with the address fields in order to cause messages to be misdelivered, the result would not be a useful association.

The checksum field is computed by a algorithm using all the bits in the message including the address fields in the message header, then is ordinarily encrypted along with the sequence-number field by an appropriate algorithm using the specified key, so that the intended receiver is assured only the intended sender could have generated it. In the Internet system, the natural choice for checksum is the 16-bit, ones-complement algorithm [6], while the natural choice for encryption is the DES algorithm [4] (see the discussion following for further consideration on these points). The detailed procedures are as follows:

1. The requestor  $i$  rolls a random message ID  $I$  and sends it and the association specifier  $[A(i),A(j)]$  as a request to the Cookie Jar. The message header includes the addresses  $[A(i),A(C)]$ , where  $A(C)$  is the address of the Cookie Jar. The following schematic illustrates the result:

|               |  |                 |
|---------------|--|-----------------|
| +-----+-----+ |  |                 |
| A(i)   A(C)   |  | message header  |
| +-----+-----+ |  |                 |
| I   checksum  |  | message ID      |
| +-----+-----+ |  |                 |
| A(i)   A(j)   |  | assoc specifier |
| +-----+-----+ |  |                 |

2. The Cookie Jar checks the access list to determine if the association  $[A(i),A(j)]$  is valid. If so, it rolls a random number  $K(i,j)$  and constructs the reply below. It checksums the message,

encrypts the  $j$  cookie field with  $K(j)$ , then encrypts it and the other fields indicated with  $K(i)$  and finally sends the reply:

|               |  |                                |
|---------------|--|--------------------------------|
| +-----+-----+ |  |                                |
| A(C)   A(i)   |  | message header                 |
| +-----+-----+ |  |                                |
| I   checksum  |  | message ID (encrypt $K(i)$ )   |
| +-----+-----+ |  |                                |
| K(i,j)        |  | i cookie (encrypt $K(i)$ )     |
| +-----+       |  |                                |
| K(i,j)        |  | j cookie (encrypt $K(j)K(i)$ ) |
| +-----+       |  |                                |

3. Upon receipt of the reply the requestor  $i$  decrypts the indicated fields, saves the (encrypted)  $j$  cookie field and copies the  $i$  cookie field to the  $j$  cookie field, so that both cookie fields are now the original  $K(i,j)$  provided by the Cookie Jar. Then it verifies the checksum and matches the message ID with its list of outstanding requests, retaining  $K(i,j)$  for its own use. It then rolls a random number  $X$  for the  $j$  cookie field (to confuse wiretappers) and another  $I'$  for the (initial) message ID, then recomputes the checksum. Finally, it inserts the saved  $j$  cookie field in the  $i$  cookie field, encrypts the message ID fields with  $K(i,j)$  and sends the following message to its associate:

|               |  |                                |
|---------------|--|--------------------------------|
| +-----+-----+ |  |                                |
| A(i)   A(j)   |  | message header                 |
| +-----+-----+ |  |                                |
| I'   checksum |  | message ID (encrypt $K(i,j)$ ) |
| +-----+-----+ |  |                                |
| K(i,j)        |  | i cookie (encrypt $K(j)$ )     |
| +-----+       |  |                                |
| X             |  | j cookie (noise)               |
| +-----+       |  |                                |

4. Upon receipt of the above message the associate  $j$  decrypts the  $i$  cookie field, uses it to decrypt the message ID fields and verifies the checksum, retaining the  $I'$  and  $K(i,j)$  for later use. Finally, it rolls a random number  $J'$  as its own initial message ID, inserts it and the checksum in the confirm message, encrypts the message ID fields with  $K(i,j)$  and sends the message:

|               |  |                                |
|---------------|--|--------------------------------|
| +-----+-----+ |  |                                |
| A(j)   A(i)   |  | message header                 |
| +-----+-----+ |  |                                |
| J'   checksum |  | message ID (encrypt $K(i,j)$ ) |
| +-----+-----+ |  |                                |

5. Subsequent messages are all coded in the same way. As new data are generated the message ID is incremented, a new checksum computed and the message ID fields encrypted with  $K(i,j)$ . The receiver decrypts the message ID fields with  $K(i,j)$  and discards the message in case of incorrect checksum or sequence number.

### 3. Discussion

Since the access lists are considered public read-only, there is no need to validate Cookie Jar requests. A perpetrator might intercept, modify and replay portions of Cookie Jar replies or subsequent messages exchanged between the associates. However, presuming the perpetrator does not know the keys involved, tampered messages would fail the checksum test and be discarded.

The "natural" selection of Internet checksum algorithm and DES encryption should be reconsidered. The Internet checksum has several well-known vulnerabilities, including invariance to word order and byte swap. In addition, the checksum field itself is only sixteen bits wide, so a determined perpetrator might be able to discover the key simply by examining all possible permutations of the checksum field. However, the procedures proposed herein are not intended to compensate for weaknesses of the checksum algorithm, since this vulnerability exists whether authentication is used or not. Also note that the encrypted fields include the sequence number as well as the checksum. In EGP and the proposed DGP the sequence number is a sixteen-bit quantity and increments for each successive message, which makes tampering even more difficult.

In the intended application to EGP, DGP and similar protocols associations may have an indefinite lifetime, although messages may be sent between associates on a relatively infrequent basis. Therefore, every association should be rekeyed occasionally, which can be done by either associate simply by sending a new request to the Cookie Jar and following the above procedure. To protect against stockpiling private user keys, each user should be rekeyed occasionally, which is equivalent to changing passwords. The mechanisms for doing this are beyond the scope of this proposal.

It is a feature of this scheme that the private user keys are not disclosed, except to the Cookie Jar. This is why two cookies are used, one for  $i$ , which only it can decrypt, and the other for  $j$ , decrypted first by  $i$  and then by  $j$ , which only then is valid. An interceptor posing as  $i$  and playing back the Cookie Jar reply to  $j$  will be caught, since the message will fail the checksum test. A perpetrator with access to both the Cookie Jar reply to  $i$  and the subsequent message to  $j$  will see essentially a random permutation of

all fields. In all except the first message to the Cookie Jar, the checksum field is encrypted, which makes it difficult to recover the original contents of the cookie fields before encryption by exploiting the properties of the checksum algorithm itself.

The fact that the addresses in the message headers are included in the checksum protects against playbacks with modified addresses. However, it may still be possible to destabilize an association by playing back unmodified messages from prior associations. There are several possibilities:

1. Replays of the Cookie Jar messages 1 and 2 are unlikely to cause damage, since the requestor matches both the addresses and once-only sequence number with its list of pending requests.
2. Replays of message 3 may cause user j to falsely assume a new association. User j will return message 4 encrypted with the assumed session key, which might be an old one or even a currently valid one, but with invalid sequence number. Either way, user i will ignore message 4.
3. Replays of message 4 or subsequent messages are unlikely to cause damage, since the sequence check will eliminate them.

The second point above represents an issue of legitimate concern, since a determined attacker may stockpile message 3 interceptions and replay them at speed. While the attack is unlikely to succeed in establishing a working association, it might produce frequent timeouts and result in denial of service. In the Needham-Shroeder scheme this problem is avoided by requiring an additional challenge involving a message sent by user j and a reply sent by user i, which amounts to a three-way handshake.

However, even if a three-way handshake were used, the additional protocol overhead induced by a determined attacker may still result in denial of service; moreover, the protocol model is inherently resistant to poor network performance, which has the same performance signature as the attacker. The conclusion is that the additional expense and overhead of a three-way handshake is unjustified.

#### 4. Application to EGP and DGP

This scheme can be incorporated in the Exterior Gateway Protocol (EGP) [2] and Dissimilar Gateway Protocol (DGP) [3] models by adding the fields above to the Request and Confirm messages in a straightforward way. An example of how this might be done is given in [3]. In order to retain the correctness of the state machine, it is

convenient to treat the Cookie Jar reply as a Start event, with the understanding that the Cookie Jar request represents an extrinsic event which evokes that response.

The neighbor-acquisition strategy intended in the Dissimilar Gateway Protocol DGP follows the strategy in EGP. The stability of the EGP state machine, used with minor modifications by DGP, was verified by state simulation and discussed in an appendix to [2]. Either associate can send a Request command at any time, which causes both the sender and the receiver to reinitialize all state information and send a Confirm response. In DGP the Request operation involves the Cookie Jar transaction (messages 1 and 2) and then the Request command itself (message 3). In DGP the keys are reinitialized as well and each retransmission of a Request command is separately authenticated.

In DGP the Request command (message 3) and all subsequent message exchanges assume the keys provided by the Cookie Jar. Use of any other keys results in checksum discrepancies and discarded messages. Thus the sender knows its command has been effected, at least at the time the response was sent. If either associate lost its state variables after that time, it would ignore subsequent messages and it (or its associate) would eventually time out and reinitiate the whole procedure.

If both associates attempt to authenticate at the same time, they may wind up with the authentication sequences crossing in the network. Note that the Request message is self-authenticating, so that if a Request command is received by an associate before the Confirm response to an earlier Request command sent by that associate, the keys would be reset. Thus when the subsequent Confirm response does arrive, it will be disregarded and the Request command resent following timeout. The race that results can only be broken when, due to staggered timeouts, the sequences do not cross in the network. This is a little more complicated than EGP and does imply that more attention must be paid to the timeouts.

A reliable dis-association is a slippery concept, as example TCP and its closing sequences. However, the protocol model here is much less demanding. The usual way an EGP association is dissolved is when one associate sends a Cease command to the other, which then sends a Cease-ack response; however, this is specifically assumed a non-reliable transaction, with timeouts specified to break retry loops. In any case, a new Request command will erase all history and result in a new association as described above.

Other than the above, the only way to reliably dis-associate is by timeout. In this protocol model the associates engage in a

reachability protocol, which requires each to send a message to the other from time to time. Each associate individually times out after a period when no messages are heard from the other.

## 5. Acknowledgments

Dan Nasset and Phil Karn both provided valuable ideas and comments on early drafts of this report. Steve Kent and Dennis Perry both provided valuable advice on its review strategy.

## 6. References

- [1] Kent, S.T., "Encryption-Based Protection for Interactive User/Computer Communication", Proc. Fifth Data Communications Symposium, September 1977.
- [2] Mills, D.L., "Exterior Gateway Protocol Formal Specification", DARPA Network Working Group Report RFC-904, M/A-COM Linkabit, April 1984.
- [3] Mills, D.L., "Dissimilar Gateway Protocol Draft Specification", in preparation, University of Delaware.
- [4] National Bureau of Standards, "Data Encryption Standard", Federal Information Processing Standards Publication 46, January 1977.
- [5] Needham, R.M., and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers", Communications of the ACM, Vol. 21, No. 12, pp. 993-999, December 1978.
- [6] Postel, J., "Internet Protocol", DARPA Network Working Group Report RFC-791, USC Information Sciences Institute, September 1981.