# An Authentication and Key Distribution System for Open Network Systems

Shiuh-Pyng Shieh, Wen-Her Yang

**Abstract**

*In this paper, we present a four-message protocol for initial authentication that supports subsequent authentication in two messages without contacting the authentication server or using synchronized clocks. This protocol has the property of using fewer messages to provide better security. Based on the protocol, a secure authentication system is built, which uses the master/slave scheme to provide better reliability and reduce the load on an authentication server.*

*Index Terms – Distributed system security, network security*

## 1. Introduction

Timestamps are now used in many production authentication systems such as Kerberos [Steiner88] [Kohl93]. Kerberos is based on the Needham and Schroeder secret key protocol [Needham78] and uses timestamps suggested by Denning and Sacco [Denning81]. It was developed at MIT [Steiner88] to provide a range of authentication and security facilities for the use in open network systems, where neither the trustworthiness of clients nor the security of the network and machines offering network services can be assumed. Timestamps depending on reliable synchronized clocks are used to assure the freshness of messages. As Gong [Gong92] discusses the difficulty of recovering from a post-dated clock, an authenticator can be replayed when the time incorrectly recorded in the authenticator is reached. Kerberos is vulnerable to replay attacks [Bellovin90] if the client and the server clocks are not at least loosely synchronized, which means an intruder can replay an old (invalid) authenticator and an old ticket to open a new channel with the server, then replay old data messages cached. If a synchronization protocol is used to bring client and server clocks into loose synchrony, the synchronization protocol itself must be secure against security attacks. Therefore, the application of Kerberos is somewhat limited.

Kehne et al. [Kehne92] present a protocol (KSL) for multiple authentications that serve as a nonce based alternative to Kerberos. The KSL protocol is a five-message protocol for initial authentication that supports subsequent authentication in three messages. Otway and Rees [Otway87] present a four-message protocol that only supports initial authentication. Neuman et al. [Neuman93] present a nonce-based protocol (NS) for key distribution that contains four messages in the initial exchange and three messages for subsequent exchanges. Both the NS protocol and Otway protocol are vulnerable to substitution attacks in some implementations, therefore Syverson [Syverson93] proposes a permuted protocol for improvement. The Permuted protocol is basically the same as the NS protocol. It simply switches the order of the variables in the authentication messages to avoid substitution attacks.

This work was supported in part by the National Science Council of Taiwan under contract NSC-82-0408-E-009-290.
UNIX is the trademark of AT&T. SunOS is the trademark of SUN Microsystem. DOS is the trademark of Microsoft Corp.
The authors are with the Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan 30050.

Both the NS protocol and the Permuted protocols has the weakness that the loss of authentication messages may be undetected and legal service requests may be denied. In these protocols, the client is not able to determine whether or not the server receives the session key at the end of the initial authentication. The uncertainty limits the use of these protocols in client−server architectures. Considering the case, at the end of the initial authentication, the fourth message in these protocols is lost, which contains the session key to be transmitted from the client to the server. Thus, the client possesses the session key, but the server does not. The loss of message cannot be detected by the client, because the client cannot obtain the belief that the server has received the message which contains the session key [Neuman93]. If the client starts transmitting data message on the communication channel, the server may consider the first data message he receives from the client as the fourth message of the protocol, and verify its correctness and freshness. As a result, the verification fails and the server considers the authentication request representing an attack from an intruder, and these messages should be discarded. Thus, legal service requests are denied. On the other hand, the client thought all data had been successfully transmitted. The loss of data remains undetected by both parties, and may cause unrecoverable errors in some applications. The NS and Permuted protocols need additional messages or supports to deal with the faults.

KryptoKnight [Molva92] is an authentication and key distribution system designed by IBM. It is a symmetric secret−key protocol and uses either DES or MD5. It needs seven messages for initial authentication and three messages for subsequent authentications. Its ticket contains a session key with the expiration time to be shared by both the client and the server. As a result, KryptoKnight is also vulnerable to the replay attack as in Kerberos. It is possible to replay an old (invalid) ticket in KryptoKnight if system clocks are not synchronized. Among all these protocols described above, Kerberos and KryptoKnight have been widely or commericially implemented.

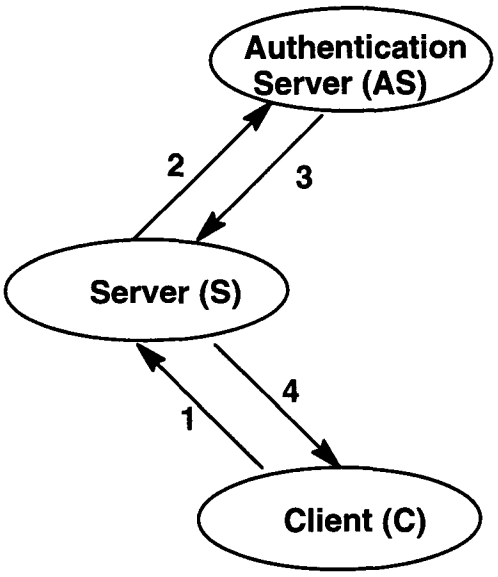
As an improvement, we propose a new secure authentication protocol, which uses symmetric cryptography and has the property of using fewer messages to achieve the goals of authentication and key distribution without the use of synchronized clocks. This protocol needs four authentication messages protocol for initial authentication, and two messages for subsequent authentication. Based on this protocol, an authentication system is built, and a number of secure network protocols (*rlogin, telnet, rsh* and *ftp*) have been ported in this environment. The authentication system has been running on campus since March 1994, and currently supports SunOS, Solaris, DOS, Linux, NetBSD and FreeBSD.

This paper is organized as follows. In section 2, a new authentication protocol for both initial and subsequent authentications is proposed. In sections 3 and 4, the authentication system is overviewed and the comparisons with other protocols are provided. Finally, conclusions and formal proofs of our protocol are given in section 5 and appendix, respectively.

## 2. The Authentication Protocol

In the previous section, we discussed the weakness of a number of protocols. In this section, a new authentication protocol will be presented as an improvement. This protocol has the property of using fewer messages to provide better security. In our protocol, we use nonces (which are implemented as random numbers in our system) instead of timestamps as in Kerberos. The protocol for initial authentication shown in Figure 1 consists of four steps:

1) $C \rightarrow S$: $C, \{C, N_c\}_{Kc}$
   In order to request service from a server S (e.g., *rlogind, telnetd*), the client generates a nonce (a random number) $N_c$, and sends it to the server together with his identity C, both encrypted with his secret key (password) $K_c$. $N_c$ is used to detect the replay of old messages.




1. Request for a service

   $C \rightarrow S$: C, $\{C, N_c\}_{Kc}$

2. Request for Authentication

   $S \rightarrow AS$: S, $\{S, N_s, C, \{C, N_c\}_{Kc}\}_{Ks}$

3. Session key for the server

   $AS \rightarrow S$: $\{AS, K_{cs}, (N_s+1), \{(N_c+1), K_{cs}\}_{Kc}\}_{Ks}$

4. Session key and ticket for the client

   $S \rightarrow C$: $\{(N_c+1), K_{cs}\}_{Kc}$, $\{C, K_{cs}, T_s\}_{Ks}$

Figure 1. New Secure Authentication Protocol

C and $N_c$ are encrypted to prevent forged message attacks.

2) $S \rightarrow AS$: S, $\{S, N_s, C, \{C, N_c\}_{Kc}\}_{Ks}$
Upon receiving the service request from the client, the server needs to determine the identity of the client. If the server checks his access control list and is willing to serve the client, it generates a nonce $N_s$, and sends it to the authentication server (AS) together with his identity S, client's identity C, and the request message from the client $\{C, N_c\}_{Kc}$, all encrypted with the server secret key $K_s$. $N_s$ is used to detect the replay of old messages. Since the access control list in UNIX−like systems is built in the server, the server can reject the service request if the client is not authorized to access. Thus, unnecessary authentication steps can be avoided. In contrast, all principals in Kerberos directly communicate with authentication servers. Therefore, all network protocols (*rlogin, telnet* and *ftp*) that Kerberos supports in UNIX−like systems needs to transmit four additional messages before the server rejects an illegal request. Further details about the design of these network protocols in our authentication system can be found in section 3.1.

3) $AS \rightarrow S$: $\{AS, K_{cs}, (N_s+1), \{(N_c+1), K_{cs}\}_{Kc}\}_{Ks}$
Upon receiving the request message for authentication from the server, the authentication server AS queries his authentication database and finds corresponding server key ($K_s$) as well as the the client password ($K_c$). Using these two keys, the authentication server can decrypt the received message, and derive S, C, $N_c$ and $N_s$. Then the authentication server increments both $N_c$ and $N_s$ by 1, and generates a new session key ($K_{cs}$). $N_c$, $N_s$ and $K_{cs}$ are all encrypted with the secret key $K_s$, and sent back to the server.

4) $S \rightarrow C$: $\{(N_c+1), K_{cs}\}_{Kc}$, $\{C, K_{cs}, T_s\}_{Ks}$
Upon receiving the message from the authentication server, the server uses his secret key to decrypt the message and gets the session key, $(N_s+1)$, as well as $\{K_{cs}, (N_c+1)\}_{Kc}$. The server checks if the returned nonce is equal to

1'. $C \rightarrow S$: $\{C, N_c'\}_{Kcs}$, $\{C, K_{cs}, T_s\}_{Ks}$,

2'. $S \rightarrow C$: $\{N_c'+1, K_{cs}'\}_{Kcs}$

Figure 2. Subsequent Authentication

($N_s+1$). If yes, it will regard the message as a replay of old messages. If not, the server generates a ticket $\{c, K_{cs}, T_s\}_{Ks}$, and forwards it and $\{K_{cs}, (N_c+1)\}_{Kc}$ to the client. The ticket can be used by the client for subsequent communication with the server, and $T_s$ is the expiration time of the ticket. The client can determine the freshness of the message by checking the returned nonce ($N_c+1$) in the reply message. If the message is fresh, the client uses the received session key $K_{cs}$ to encrypt succeeding data messages in the session. The ticket and the session key will be stored for subsequent authentication. In both Kerberos and KryptoKnight, it is possible to replay an old (invalid) ticket if system clocks are not synchronized. Our protocol can detect these replay attacks because the server S determines the validity of a ticket by comparing its local clock with $T_s$, which was also generated on the same server machine.

In case of failure, if message 4 is lost and the timer expires, the client will close the old communication channel*, open a new channel, and issue a new request to the server. The new request will be transmitted over the new channel without causing any confusion. In this way, the client will not transmit data messages unless he is assured that the authentication has been successfully completed. Therefore, the succeeding data messages will not be mixed up with message 4 of our protocol as those appeared in the NS and Permuted protocols.

When the initial authentication of our protocol completes, both the client and the server acquire the session key; the client is certain that the server has received the session key, but the server is not certain whether or not the client has received the session key. However, these beliefs are already sufficient for mutual authentication, because the client is the one who initiates the service request and sends succeeding data messages. It is possible to acquires more belief when the server receives the data messages (such as user name, password in *telnet* protocol) from the client. If the server can successfully decrypt the data messages with the session key, he is certain that the client has successfully received the session key. The beliefs that can be derived from our protocol can be analyzed in the formal way. All details can be found in the appendix.

## 2.1 Subsequent Authentication

After the initial authentication, the client possesses a ticket and the session key that can be used for subsequent authentications. In the subsequent protocol (See Figure 2), the client simply sends the server the ticket $\{C, K_{cs}, T_s\}_{Ks}$ along with $\{c, N_c'\}_{Kcs}$. $\{c, N_c'\}_{Kcs}$ is used to prevent forged message attacks. Upon receiving the request message from the client, the server checks the validity of the ticket by comparing $T_s$ with his local clock. Note that the expiration time $T_s$ was also generated on the same server machine. If $T_s$ is larger than the local clock (that is, the ticket is valid), the server generates a new session key $K_{cs}'$ and sends a reply $\{N_c'+1, K_{cs}'\}_{Kcs}$ back to the client. A new session key $K_{cs}'$ will be generated for each subsequent authentication, and only used in a session (Here, we assume that each server is capable of generating good session keys). Since $K_{cs}'$ will not

* A channel is implemented as a pair of sockets, the client socket and the server socket, in UNIX systems. If any socket of the pair is closed, the channel is closed.

be reused, replay attacks can be prevented. The client can determine the freshness of the reply by checking the value of received $N_c$'. At the end of the subsequent authentication, the client encrypts the succeeding data messages in this session with the new session key $K_{cs}$' and sends them to the server. Upon receipt of the first data message, the server can determine the freshness of this request. The use of the new session key ensures the freshness of the request and avoids the replay of old messages, thus provides better security.

## 3. Design and Implementation

The authentication system is built based on the proposed authentication protocol. A set of libraries is provided for use by client applications and services. The DES encryption algorithm is used, but it is implemented as a separate module that can be easily replaced. Our DES codes runs at the speed of 8 Mbits/sec on SUN Sparc 20. In our system, the master–slave scheme is used to provide better reliability and reduce the load on an authentication server. That is, It is possible to have one master authentication server and several slave authentication servers, each of which has a copy of the same authentication database. Updates are applied to both the master and slave copies simultaneously to keep the consistency between the master database and the slave databases. In contrast, in Kerberos, updates are initially applied only to the master database, and then the master server broadcast the changes hourly to the slave servers. Before the broadcast, the authentication databases are inconsistent.

### 3.1 Secure *telnet*

The authentication system currently supports four internet services: *rlogin*, *telnet*, *rsh*, and *ftp*. Because of their similarity, only *telnet* will be illustrated herein. When users request telnet services and enter their username and password, the client process (*telnet*) on the local machine will send a connection request to server process (*telnetd*) on the remote machine. After receiving the request, *telnetd* will check his access control list to determine whether or not the client is authorized to login the machine. If the check succeeds, *telnetd* will request the authentication server (AS) to authenticate the client using the new protocol proposed in section 2. If the authentication is successful, AS will generate a session key and reply to *telnetd*. When *telnetd* receives the session key, it forwards the session key to the client *telnet*. The *telnet* protocol is depicted in Figure 3.

### 3.2 Master/Slave Authentication Servers

In order to balance the load on an authentication server, our authentication system uses a master/slave architecture as shown in Figure 4. The authentication system has a master machine and a number of slave machines. The authentication services can be performed either on the master machine or on any slave machine. The master machine contains the master copy of the authentication database, whereas the slave machine contains the read–only copy of the authentication database. The replication of the database has the advantages of higher availability and better performance. Each authentication database contains user IDs and their secret keys. Updates will be immediately applied to both the master machine and all slave machines. In the Master/Slave architecture, when a client sends a login request to a server, the server will choose an AS with highest priority and send him a request to authenticate the client. In the system we design, each server keeps a configuration file containing the list of all ASs. For example, the AS list on server 3 in Figure 5 is as follows:

| | |
|---|---|
| Slave | Slave_AS1 |
| Slave | Slave_AS2 |
| Master | Master_AS |
| Slave | Slave_AS3 |
| : | : |
| Slave | Slave_AS9 |

where Slave_AS1 has the highest priority and Slave_AS9 has the lowest. The priority of an AS in

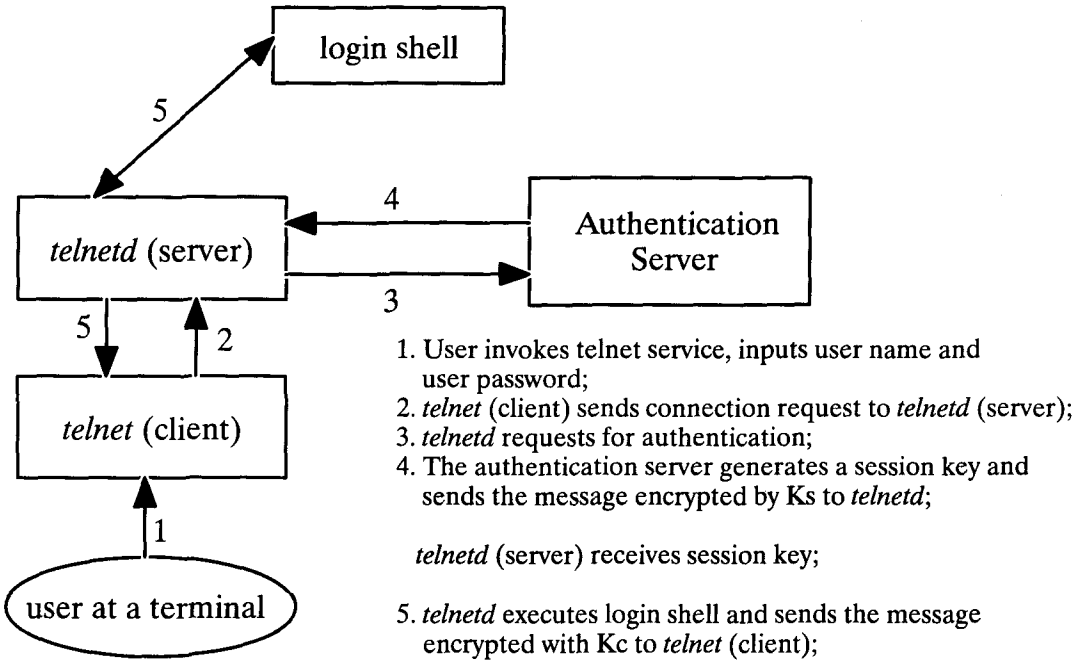



Figure 3. Secure *telnet* Protocol

the AS list on a server machine depends on the distance from the server to the AS, the load on the AS, etc. A nearer or light−load AS should get higher priority such that servers can get quicker response from the AS. In case of failure, the active AS with higher priority will be requested. The authentication system will be able to function well unless all authentication servers fails. Therefore, the system has higher availability and better performance.

We started the design and implementation of the authentication system in January, 1993, and the prototype has been running and tested on many workstations and PCs since March 1994. Now it has been widely used on campus. The authentication system currently supports PCs and workstations running SunOS, Solaris, DOS, Linux, NetBSD and FreeBSD. Several popular secure communication protocols *rlogin*, *telnet* and *ftp* have been ported. From our experience with the authentication system, users do not feel its existence. In order to monitor the network traffic, we also designed and implemented a network monitor tool. This tool can capture and read any message in a network that the administrator is interested in. People who are interested in further details regarding the authentication system and the network monitor tool can find the binary codes and the installation guide in the directory /pub/CSIE/Shield by ftp ftp.csie.nctu.edu.tw.

## 4. Comparisons

Among all the protocols that are described above, our protocol has the following characteristics.

(1) It needs minimal number of messages for initial and subsequent key exchanges.

(2) It can prevent the substitution attacks, and malicious replay of old messages.

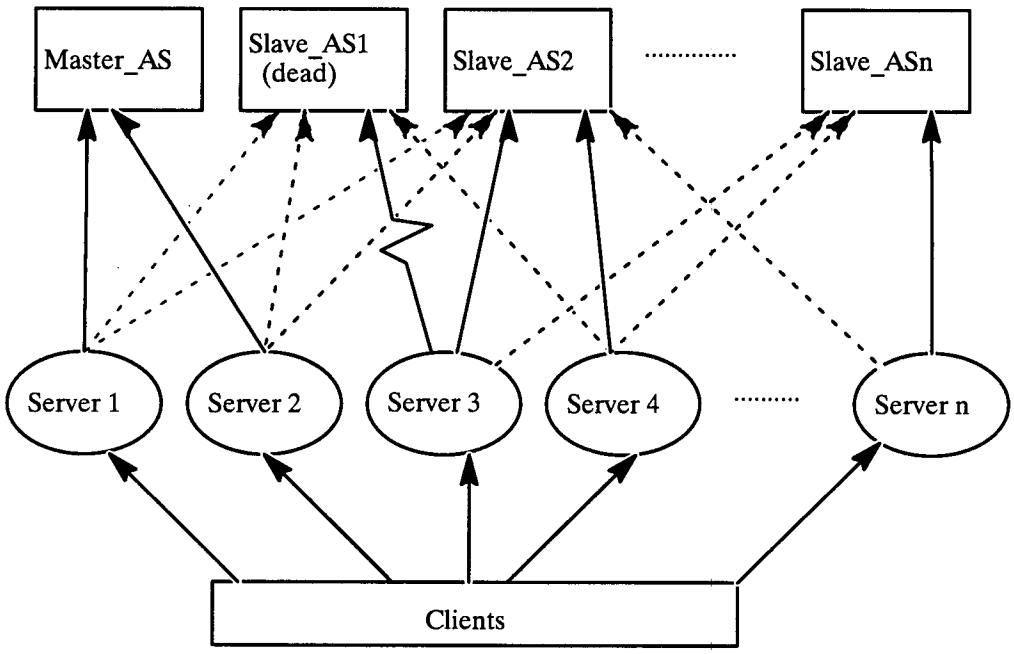



Figure 4. Authentication Requests in new protocol

(3) It can detect the loss of authentication messages, and therefore legal service requests will not be denied.

(4) It can avoid unnecessary authentication steps before rejecting an illegal request.

(5) Based on the protocol, An actual authentication system has been designed, implemented and tested on UNIX−like systems.

(6) The authentication databases on the authentication server machines are kept consistent all the time, if all of the authentication servers are active.

These comparisons are summarized in Table 1.

| | # of messages for initial authentication | # of messages for subsequent authentication | Limitations |
|---|---|---|---|
| Kerberos | 6 | 2 | replay attack |
| KryptoKnight | 7 | 3 | replay attack |
| Otway−Rees | 4 | 4 | substitution attack |
| NS protocol | 4 | 3 | substitution attack faults undetected |
| Permuted protocol | 4 | 3 | faults undetected |
| KSL protocol | 5 | 3 | no |
| Our protocol | 4 | 2 | no |

Table 1. Comparison with other protocols

## 5. Conclusions

In our authentication system, authentication servers are isolated from clients. Only registered servers, e.g, *telnetd*, can directly communicate with the authentication servers, and clients have to communicate with authentication servers through registered servers. This protocol matchs the current design of UNIX−like systems, where the access control lists is built in the servers. With this protocol, illegal service requests from principals can be rejected by the registered servers before they are forwarded to authentication servers. Thus, unnecessary authentication steps can be avoided and the load on the authentication server can be reduced. Our authentication protocol needs fewer messages for initial and subsequent authentications, while providing strong beliefs. Not only does it ensure the security, but also strengthen the reliability of open network systems.

## Acknowledgement

We are grateful to Hsiang−Ting Cheng, Fu−Shen Ho, and Yu−Lun Huang of National Chiao Tung University for their continuing efforts in implementing the authentication system.

## References

[Bellovin90] S.M. Bellovin, M. Merritt, "Limitations of the Kerberos Authentication System," ACM Communications Review, vol. 20, no. 5, pp. 119−132, October 1990.

[Burrows90] M. Burrows, M. Abadi, R. Needham, "A Logic of Authentication," *ACM Transactions on Computer Systems*, vol. 8, No. 1, February 1990, pp 18−36.

[Denning81] D. E. Denning, G. M. Sacco, "Timestamps in Key Distribution Protocols," *Communication of the ACM*, vol. 24, no. 8, pp 533−536, August 1981.

[Gong92] L. Gong, "A Security Risk of Depending on Synchronized Clocks, *ACM Operating System Review*, 26(1), 1992.

[Kehne92] A. Kehne, J. Schonwalder, H. Langendorfer, "A Nonce−Based Protocol for Multiple Authentications," *ACM Operating Systems Review*, 26(4):84−89, October 1992.

[Kohl93] J. Kohl, C Neuman, "The Kerberos network authentication service (V5)," *Internet RFC 1510*, Sept. 1993.

[Molva92] R. Molva, G. Tsudik, E. V. Herreweghen and S. Zatti, "KryptoKnight Authentication and Key Distribution System," *Proc. ESORICS '92*, Nov. 1992.

[Needham78] Roger M. Needham, Michael D. Schroeder, "Using encryption for authentication in large networks of computers," *Communication of the ACM*, 21(12): 993−999, December 1978.

[Neuman93] B. Clifford Neuman, Stuart G. Stubblebine, "A Note on the Use of Timestamps as Nonces," *ACM Operating Systems Review*, pp 10−14, April 1993.

[Otway87] D. Otway, O. Rees, "Efficient and Timely Mutual Authentication," *ACM Operating System Review*, vol 21, no. 1, pp. 8−10, 1987.

[Steiner88] J.G. Steiner, B.C. Neuman, J.I. Schiller, "Kerberos: An Authentication service for Open Network Systems," *Proceedings of the Winter 1988 Usenix Conference,* pp 191−201, February 1988.

[Syverson93] Paul Syverson, "On Key Distribution Protocols for Repeated Authentication," *ACM Operating Systems Review*, pp 10−14, October, 1993.

## Appendix. Protocol Analysis

In this appendix, we explain why our protocol can reach the goals of authentication with fewer messages for both initial and subsequent authentications.

## A.1 Initial Authentication

We use a formal method presented by Burrows, Abadi and Needham [Burrows90]. As Burrows et al. state that authentication is complete between two parties, the client and the server, if there is a $K_{cs}$ such that

$$C \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S, \quad S \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S.$$

These two formulas state that each principal trusts $K_{cs}$ as a secret session key shared with the other one. Some protocols may provide more beliefs:

$$C \textbf{ believes } S \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S, \quad S \textbf{ believes } C \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S.$$

These two formulas state that each principal believes that the other one currently trusts the session key. The functional objectives of the initial protocol are to prove (1) the presence of both parties to each other, and (2) the client's receipt of a ticket and a session key for subsequent authentication. Both the NS and Permuted protocols only satisfy the following reduced set of formalized goals to achieve the functional objectives:

$$C \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S,$$

$$S \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S,$$

$$S \textbf{ believes } C \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S$$

This set of formalized goals is sufficient for the completion of authentication. However, protocols that merely achieve this set of goals may have the weakness that faults may be undetected and legal service requests may be denied, as we pointed out in section 1. In contrast, our protocol at the end of authentication can achieve the following set of formalized goals:

$$C \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S, \tag{1}$$

$$S \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S, \tag{2}$$

$$C \textbf{ believes } S \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S \tag{3}$$

The protocol achieving this set of goals will not have undetected faults and denial of services due to loss of the fourth message. In addition to the three formalized goals, our protocol can achieve one more formalized goal when the server receives the first data message X from the client:

$$S \textbf{ believes } C \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S \tag{4}$$

The proof is given as follows. When the server receives the message X, the annotation rules yield that

$$S \textbf{ sees } \{X, C \overset{K_{cs}}{\leftrightarrow} S\}_{Kcs}$$

Since we have formula (2)

$$S \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S,$$

the message−meaning rule for shared keys applies and yields the following:

$$S \textbf{ believes } C \textbf{ said } (X, C \overset{K_{cs}}{\leftrightarrow} S)$$

Since message X has not been sent and $K_{cs}$ has not been used at any time before the current run of the protocol, we have the following hypothesis:

$$S \textbf{ believes fresh}(X, C \overset{K_{cs}}{\leftrightarrow} S)$$

The nonce−verification rule applies and yields

$$S \textbf{ believes } C \textbf{ believes } (X, C \overset{K_{cs}}{\leftrightarrow} S)$$

We break a conjunction, to obtain the following:

$$S \textbf{ believes } C \textbf{ believes } C \overset{K_{cs}}{\leftrightarrow} S$$

## A.2 Subsequent Authentication

Upon receipt of message 1', S can decrypt this message to check $T_s$. Since $T_s$ has not expired, $K_{cs}$ is still good. Applying the message–meaning rule, the nonce–verification rule, and the jurisdiction rule, we get

S **believes** C $\overset{K_{cs}}{\leftrightarrow}$ S,

The server generates a new session key $K_{cs}$' for each subsequent authentication request, which means that $K_{cs}$' will not be reused, thus replay attacks can be prevented. The jurisdiction rule applies and yields

S **believes** C $\overset{K_{cs}'}{\leftrightarrow}$ S,

Upon receiving message 2', C can decrypt the message and get the session key for subsequent communication. We have

C **believes** S **believes** C $\overset{K_{cs}'}{\leftrightarrow}$ S

The jurisdiction rule applies and yields

C **believes** C $\overset{K_{cs}'}{\leftrightarrow}$ S,

Again, for the same reason as the initial protocol we described above, the server believes, when he receives the first data message from the client, that the client has received the new session key $K_{cs}$'. Thus, we have

S **believes** C **believes** C $\overset{K_{cs}'}{\leftrightarrow}$ S

Therefore, our protocol achieves the set of four formalized goals that is sufficient for the completion of reauthentication.