Jerry

M.I.T. LABORATORY FOR COMPUTER SCIENCE
Computer Systems Research Division

March 21, 1980
Request for Comments No. 183

The Authentication Server: Progress and Plans

by

D. Daniels, J. Lucassen, W. Rubin, and I. Greif

Abstract

This is a report on the progress since September, 1979 of the UROP project on the design and implementation of an Authentication Server for a distributed system. Included as well are our plans for the spring term and a request that potential users begin to tell us what services they would like from such a server.

1. Introduction

The authentication server project has two goals. The first is to build a key distribution center that can be used to support other distributed system components that are to be built here. In particular, the data storage server [Svobodova, 1980] will store data in an encrypted form and will therefore require such a server. Also, any secure conversation between processes in the system might require similar services. The main function of the authentication server will be to provide for key distribution.

A second purpose of the project is to provide a source of experience with programming for a distributed system. The currently available "extended" CLU [Herlihy, 1979] will serve as a language for several experimental implementations. By reviewing our programming experiences regularly we will begin to develop some insight into how such a language can support the implementation of programs for a distributed environment.

2. Progress Report

We began meeting in September 1979, before any other projects had developed detailed specifications of their authentication server requirements. We spent about two months reading selections from the literature on protection and encryption, as well as learning "extended" CLU. At that time we decided to implement the protocols for establishing a secure conversation as presented in [Needham and Schroeder, 1978]. There are two versions of the protocols, the first for use with conventional encryption and the second based on public key encryption. These protocols should have some relationship to protocols required by local users, but are not particularly tailored to the needs of other projects in the laboratory. Thus the main results of this exercise have been initiation into the extended CLU programming environment, production of two simple servers that

can serve as foundations upon which to build, and identification of a variety of problems not addressed in the Needham and Schroeder paper. The next phase of our work will be the development of additional protocols that will extend the functionality of our server beyond that of the server defined by Needham and Schroeder.

2.1 The Conventional Encryption Implementation

The conventional key authentication server performs three services: (1) it creates a conversation key for a user who wants to communicate with another process; (2) it writes out a signature for a user who wants to sign a message; and (3) it reads signatures to authenticate them. All functions of the server are implemented using the protocols described in [Needham and Schroeder, 1978].

The protocols are described using a simple notation. We will denote the process that initiates a conversation by A, the process with which A communicates by B, a nonce identifier by I, the conversation key by CK, the secret key of A by KA, and a characteristic value by CS. An encrypted message is enclosed in brackets with the key appearing as a superscript after the message. A message that is not encrypted is enclosed in parentheses.

When a process wants a conversation key, it sends the message (A, B, I) to the authentication server. The server replies with the message: $\{I, B, CK, \{CK, A\}^{KB}\}^{KA}$. The process A can easily check the nonce identifier against the one sent in the request, thus verifying that this is an answer to that request. Once the message is verified, A can send the encrypted block $\{CK, A\}^{KB}$ to B. No one else can read the conversation key as it is transmitted in an encrypted form. When B receives the message it begins a handshaking protocol with A, so that each can be satisfied as to the identity of the other.

When a process wants to sign a message, it sends the authentication server the message $(A, \{CS\}^{KA})$, where CS is the characteristic value of the message to be signed. The server then creates a signature block: $\{A, CS\}^{KAS}$, where KAS is the secret key of the authentication server. If a second process, B, needs to verify the signature, it sends this block along with its name, B, to the server. The server will decrypt the block and then encrypt it with KB. Then B can read the block and see if the message has the correct characteristic value.

In order to create and update the database of the server, we have implemented a program that allows some authorized person to enter and delete users from the database and to change their keys. This program currently has no security checks on its invocation.

2.2 The Public Key Implementation

The public key authentication server performs a very simple task -- it gives a public key to anyone who requests one. Under the protocols as described in [Needham and Schroeder, 1978], the information is sent out as pairs: $\{A, PKA\}^{SKAS}$, where SKAS is the secret key of the authentication server and PKA is the public key of A. Since all participants know the public key of the authentication server, anyone can decrypt this message. The purpose of this encryption is to allow the recipient of the message to check its authenticity.

Since the name-key pairs are always encrypted under SKAS, the encryption can be done when the information is added to the server's database. This will enhance the server's performance since there will be no need to actually carry out any encryption at the time of the request. An additional advantage is that the server does not need to know its own secret key during normal operation.

Since there is no secret information stored in the server's data-base, the server need be write-protected only. This simplifies the implementation and can thus add to one's confidence in the security of the server.

A working implementation of the server is available. Also provided are some programs for interface to the server:

> For those interested in sending CLU objects between virtual machines, there is the 'channel' cluster. It aids the user in setting up inter-process communication links, maintains the keys needed for a particular conversation, and enforces the integrity checks on transmitted data using encryption. In addition, opening a channel yields two integers (one for each end). These integers can be used in the ensuing conversation to detect and recover from message stream modification (e.g. using a scheme of the sort described in [Kent, 1976]).
>
> In order to create and update the <name-key>-pair database of the server, there is a program which allows some authorized person to enter and delete users from the database, to list them, and to change their keys. (The authorization is not checked in the current version.)

2.3 Some Comments on the Programs

The implementors of the public key and conventional encryption versions of the authentication server worked on their programs independently. The main organizational difference between the two implementations is in the use of ports. In the public key implementation the authentication server receives all messages on a single distinguished port.[1] Each client has one distinguished port on which it receives two kinds of messages -- responses from the authentication server and initial messages of conversations started by other clients. Each client creates one non-distinguished port per handshaking conversation. The conversation ports receive several kinds of messages, each of which is only acceptable at a certain stage of the conversation.

---

1. A distinguished port is one that is catalogued in the ECLU message server catalog.

The conventional encryption implementation uses one distinguished port in the server and one distinguished port per client. In addition, every message sent by a client contains a newly created "reply" port on which the client expects to receive a response. Thus the first of the three messages between users in the Needham-Schroeder protocols is sent to a user's distinguished port, but other messages go to newly create "reply" ports. Similarly, each response from the authentication server is received on a new port. This second organization assumes ports are cheap and relies on the creation of many ports that are used one time only. The result is that exactly one kind of message is acceptable on any given port. The distinguished ports are used only for initiating contact. The client keeps track of the mapping between ports and conversation states.

The communication between the authentication server and its clients resembles a procedure call in that each message received is immediately answered. Thus a programming language which supports remote procedure calls might simplify programming the client-server interface. However, the handshake between clients involves an exchange of messages that does not fit this pattern so that a remote procedure call would not be particularly appropriate.

3. Proposed Extensions and Modifications

There are three kinds of future work that we are considering. First, there are still parts of the current implementation of the Needham and Schroeder work that are incomplete. The encryption procedures do not implement secure encryption algorithms. Also, Needham and Schroeder suggest some modifications to their protocols that would facilitate caching of keys for reuse in future conversations. The current implementations require that the authentication server be involved each time a new conversation is started.

Second, there are issues that were outside the scope of the Needham and Schroeder paper that we can tackle. These include protocols for proceeding with a conversation once a key has been agreed upon and protocols for revoking a key once it has been compromised.

Third, we are interested in providing services that will be of use to people building other programs. For example, if the data storage server provides storage for large numbers of small objects, each under a separate key, then adequate performance may depend on its ability to get a large number of keys from the authentication server in response to a single request.

4. What Do You Need?

We are currently being kept informed of the authentication needs of the data storage server as those needs become apparent. We would like to hear as well from other potential users about how they could benefit from the availability of an authentication server.

5. References

Herlihy, M. P., Progress Report on CLU Message Passing Implementation, DSG Note 50, October, 1979.

Kent, S. T. Encryption-Based Protection Protocols for Interactive User-Computer Communication. MIT/LCS/TR-162, May, 1976, pp. 121.

Needham, R. M. and M. D. Schroeder, Using Encryption for Authentication in Large Networks of Computer, *CACM 21*, 12, December, 1978, pp. 993-997.

Svobodova, L., "Design of a Distributed Data Storage System," M.I.T. Laboratory for Computer Science. Computer Systems Group RFC 182, January, 1980.