

**DESIGNING AND IMPLEMENTING A NETWORK AUTHENTICATION SERVICE
FOR PROVIDING A SECURE COMMUNICATION CHANNEL/**

by

CHRISTOPHER P. CHANCE

B. S. E. E., University of South Carolina, 1979

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

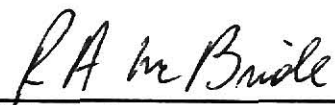
MASTER OF SCIENCE

Department of Computer Science

**KANSAS STATE UNIVERSITY
Manhattan, Kansas**

1986

Approved by:



Major Professor

LD
2668
.R4
1986
C42
c. 2

A11202 663384

Acknowledgements

I would like to dedicate this work to my wife, Kathy, and my children, Ryan and Stephanie. Without their love and understanding, this work would not have been possible. Also, I would like to thank my advisor, Dr. Rich McBride, for all of his help and direction.

TABLE OF CONTENTS

1. Introduction	1
1.1 Overview	1
1.2 Data Encryption	1
1.3 Project Overview	2
1.4 Readers Guide	3
2. Security Issues in Computer Networks	5
2.1 Security Risks	5
2.1.1 Wiretapping	5
2.1.2 Cost of Wiretapping	7
2.1.3 Accidental Threats	8
2.1.4 Trends	8
2.2 Data Security	8
2.3 Cryptography	9
2.3.1 Basic Terminology	9
2.4 Encryption Algorithms	10
2.4.1 Classical Methods	10
2.4.2 Present-day Systems	10
2.5 Conventional Encryption	11
2.5.1 The Data Encryption Standard	11
2.6 Public-key Encryption	13
2.7 Key Distribution	13
2.8 Encryption Applications in Computer Networks	14
2.8.1 Authentication Service	14
2.8.2 Network Mail	15
2.8.3 Digital Signatures	15
2.9 A Survey of Authentication Protocols and Services	15
2.9.1 Conventional and Public-key Approaches to Authentication	15
2.9.2 A Session Layer Authentication Service	16
2.9.3 An Office System Approach to Authentication	17
2.9.4 A UNIX* Secret Mail Facility	17
2.10 Comparisons and Contrasts	18

3. Design and Implementation	21
3.1 Objective	21
3.2 Design Assumptions	21
3.3 System Description	22
3.4 User Commands	23
3.4.1 Enrollx	24
3.4.2 Connectx	24
3.4.3 Sendx	25
3.4.4 Disconnectx	26
3.4.5 Getx	26
3.5 Detailed Description	27
3.5.1 Files and Data Types	27
3.5.2 Signals	28
3.5.3 Enrollx Description	28
3.5.4 Connectx and Rconx Description	30
3.5.5 Sendx and Rsendx Description	32
3.5.6 Disconnectx Description	34
3.5.7 Getx Description	35
3.5.8 The Encryption Subroutine	36
3.6 User Scenario	37
3.7 System Administration	38
4. Conclusion and Summary	40
4.1 Design Choices	40
4.2 Future Enhancements	41
4.3 Conclusion	41
REFERENCES	42

LIST OF FIGURES

Figure 1. Passive Wiretapping	6
Figure 2. Active Wiretapping	7
Figure 3. DES Iteration	12
Figure 4. Modules of the Authenticated Mail Service	23
Figure 5. User Data Structure	28
Figure 6. Message Exchange in a Typical User Session	37

1. Introduction

1.1 Overview

Authentication, within the context of computer security, is defined as the verification of data and the verification of the identity of a user or system. The verification of data insures that the data has not been tampered with either deliberately or accidentally. The purpose of this paper is to provide an overview of an authentication service that was developed to implement a secure communication channel within a network of UNIX* based machines. This service provides a "secret" electronic mail facility that verifies the identities of the communicating parties and preserves the integrity of the message contents.

Within a computer network, a user typically does not have control over the data communication lines and switching nodes that tie the network together. Unless there is a secure channel dedicated to the user, data will be transmitted over a telecommunication common carrier which is susceptible to malicious attack by an intruder. Therefore, the user cannot make any assumptions regarding the safety of the data that he puts out on the network. For this reason there exists a need for providing a secure means of communication, particularly if the data is of a sensitive nature or of value to an unfriendly user.

1.2 Data Encryption

Data encryption is a technique often used for implementing a secure channel within an insecure network. Through encryption, the data is rendered unintelligible, hopefully, to any user who does not possess the key necessary to perform the decryption process. The

* UNIX is a trademark of AT&T Bell Laboratories.

encryption of data can be done within any layer of the network architecture. With reference to the seven layer Open System Interconnection (OSI) model developed by the International Standards Organization, encryption can be done anywhere from the physical layer to the application layer [Tann81]. When done at the data link or physical layer, a data encryption/decryption device is built directly within the line interface to the network. This hardware approach could also be applied at the network or transport layers. The other choice for implementing encryption is at the user levels of a network topography. This software approach is normally done at the presentation layer or the application layer on a per session basis. The user performs the data transformation through the use of an encryption algorithm that provides the private data transmission. The subject of this paper is a software approach to providing a secure communication channel.

1.3 Project Overview

The authentication service was designed using the concept of servers [Need78]. There are three servers that comprise the system:

1. Authorization Server (AS).
2. Name Server (NS).
3. Message Server (MS).

The AS provides the user interface to the system. It enables a user to enroll in the service and establishes the protocol between communicating parties in the form of conversation keys. The encryption and decryption of messages is performed by the AS as well as providing the delivery mechanism for the messages. The NS is the interface between the registration database and the AS. The registration database consists of files containing enrollees, available machines, and user information. The major function of the NS is to process requests from the AS that require interfacing to the database. The creation of

unique conversation keys and the creation of entries in the user files are also performed by the NS. The MS provides the user interface to the "mailboxes". When a user is notified of authenticated mail, the MS is invoked to deliver the message to the intended recipient.

The scope of this paper will focus on the design and implementation of the AS and the user interface to the MS. The implementation was written in C Language¹ and was developed in a UNIX programming environment.

1.4 Readers Guide

Chapter Two of this paper continues with an in-depth look at security hazards and potential threats to a computer network. A survey of security methodologies and protocols that have been proposed or are currently in use is given. A comparison is made between different encryption techniques, and terms relating to the subject are defined. Concepts which were used in the design of this service and how they relate to the above-mentioned surveys are presented and explained.

Chapter Three covers the design, development and implementation of the service. The specific modules and their user and functional requirements will be presented with their design assumptions. The range of services will be covered, and an example of the system in use will be shown.

Chapter Four is a summary of the work done and it contains suggestions for future enhancements and extensions. Problems encountered in the project's design and justification for certain methods that were chosen are also included.

Appendices containing manual pages for the user and instructions for system

1. C language is a high level programming language developed by Bell Laboratories.

administration of the service are attached. Also included in the appendices is a copy of the code for each program within the system.

2. Security Issues in Computer Networks

2.1 Security Risks

The main interprocessor communication facility for the UNIX system and for most operating systems is a dial-up variety. The system commands "uucp" and "cu" are the general networking facilities available for providing file transfers and remote processing [Nowi82]. Both of these commands make use of the telephone system as its primary communication medium. The deliberate threats to a network of this type are wiretapping and illegal access. When attempting to implement a secure channel within a network, wiretapping should be considered as the primary threat. This is because data transmission often relies on a publicly accessible medium and therefore is the most vulnerable of the three main operations of an information system [Davi84]. In addition to transmission, the other major operations of an information system involve data processing and storage. The threat to illegal access of processing and stored data of an operating system is best managed by the access controls that are applied by each host machine. Access controls in the form of "strong" login and password procedures are the best deterrent to illegal access to a machine's stored data and processing capabilities.

2.1.1 Wiretapping In today's environment, there is a wide range of long-distance telecommunication services to choose from. Businesses may be attracted by low cost and lease from carriers whose transmission paths may be consistently predictable. This adds to the vulnerability of a computer network. If the transmission path, whether it be via cable, microwave or satellite, is known, an intruder can be selective as to which path to tap into. Communication links in a dial-up network are vulnerable to two types of wiretapping [Denn82]:

1. Passive (eavesdropping).

2. Active (tampering).

Passive wiretapping (Figure 1) occurs when an intruder intercepts a message and discloses its contents. It also enables traffic analysis or the monitoring of traffic flow within the network to take place.

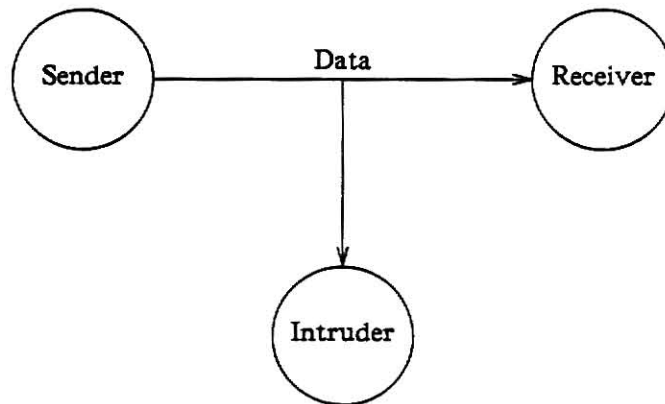


Figure 1. Passive Wiretapping

Active wiretapping (Figure 2) refers to the interception of a message by an intruder and the deliberate modification of the message contents for retransmission at a later time.

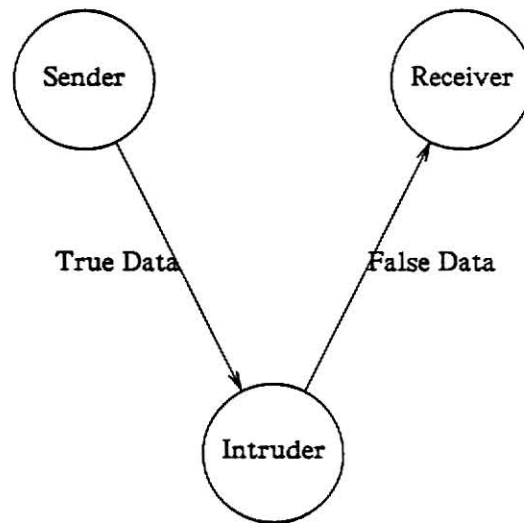


Figure 2. Active Wiretapping

2.1.2 Cost of Wiretapping Estimations of the equipment costs for tapping into the various telecommunication transmission technologies have been made by a leading manufacturer of telecommunication equipment [Gray83].

The investment for tapping into a wire line can be a few cents if it is simply a matter of strapping two telephones together on a terminal block in a wire closet. The cost can start nearing \$100 if taps of various types are placed at many points in data processing hardware or communication boxes. If the tap is placed in a trunk cable carrying multiplexed or packetized data, the investment may be as high as \$500.

For tapping into a microwave link, \$400 will allow one to buy an antenna and receiver capable of intercepting radio signals. If the signals are in complex multiplexed format, additional signal processing equipment would be necessary.

The cost for tapping into a satellite link are the highest of the three technologies. An investment of at least \$25,000 would be needed for intercepting the downlink signals. As

the complexity of the data format goes up so does the cost of interception.

From these estimations, it can be concluded that wiretapping is a very economical means for stealing data. Especially considering the fact that ever-changing technology will bring the cost of such equipment to an even lower point and to a higher degree of sophistication.

2.1.3 Accidental Threats Along with the deliberate threat of wiretapping, there is also the accidental threat of the misrouting or misdelivery of data. Due to a breakdown within a switching node in the network, sensitive data could be delivered to the wrong location.

2.1.4 Trends With today's microprocessor technology, corporate and government organizations are putting increased emphasis on the integration and expansion of telecommunication services. By expanding their telecommunication capabilities, businesses can link many dissimilar operations such as word processing, electronic mail, teleconferencing, business and engineering data processing across many locations and different computers [Kitt83]. Considering the security threats to a computer network and the increased emphasis on distributed processing, more attention is being given to the problem of insuring data privacy and authentication.

2.2 Data Security

Data security within a computer network has two primary objectives:

1. The prevention of the unauthorized disclosure of data.
2. The prevention of the unauthorized modification of data.

Security measures utilizing cryptographic systems can help meet these objectives. Encryption cannot prevent the monitoring of data but it can provide protection against disclosure and modification. Through encryption, data is rendered unintelligible to unauthorized listeners who might be tapped into the publicly accessible communication link. By encrypting data before transmission between nodes in a network, a secure

communication channel can be implemented.

The prevention of unauthorized access to data is one objective of a secure communication channel. Another important objective is to provide a method by which communicating parties can identify one another or authenticate each other. Through the use of keys, which enables the encryption and decryption process to take place, communicating users are able to identify one another and perform private communication. Unless the key has fallen into unfriendly hands, a user can be assured that the message being sent over the network will only be readable by the intended receiver who holds the proper key to decrypt the message. Encryption and message authentication is finding application in a wide range of services. Traditionally, it was a technique used almost exclusively by military and government organizations. Banking systems, medical systems, personnel records systems and systems that handle data of commercial value are currently utilizing encryption for providing secure communication. With the advent of the widespread use of electronic funds transfer (ETF) and automatic teller machines (ATM), a means for verifying transactions was necessary to protect against fraud. For this reason, the banking industry is the largest commercial user of cryptography techniques for providing data security [Davi84].

2.3 Cryptography

2.3.1 Basic Terminology Cryptography is defined as the science of devising methods of transforming data into a secret form [Prit80]. A message in its readable form is defined as plaintext or cleartext. The message after it has gone through a transformation is defined as ciphertext. The transformation process is known as encryption and the reverse process is referred to as decryption. An encryption algorithm is defined as the procedural steps for performing the transformation. A key consisting of a string of characters or bits is what

drives the algorithm for generating the unique output.

2.4 Encryption Algorithms

2.4.1 Classical Methods Basically there are three classes of ciphers:

1. Substitution ciphers
2. Transposition ciphers
3. Product ciphers

Substitution cipher is a method in which each letter or group or letters is replaced by another letter or groups of letters. The substitution can be based on a mathematical function or can use an arbitrary function initialized by a table of values.

A transposition cipher does not attempt to hide the letter or letters but simply to reorder them. Permutation is another word that describes this type of operation. The different applications of transposition ciphers range from very simple regular reordering to complex reordering in which the data is grouped into rows and columns and then is transposed based on the grouping of the data.

Present-day cryptosystems are based upon product ciphers that combine the elementary schemes of simple substitution and transposition [Lemp79]. By composing these low-cost techniques, very complex ciphers such as the Data Encryption Standard can be formed.

2.4.2 Present-day Systems In present-day ciphers, plaintext and ciphertext are considered as bit strings. Therefore, these ciphers are restricted to the binary $\{0,1\}$ alphabet due to the nature of data being stored, processed and transmitted in binary form. Stream and block encryption are two present-day methods that utilize the binary alphabet.

In stream encryption, the plaintext string is transformed bit by bit. A randomly generated bit stream is added modulo 2 to the bit stream representing the plaintext [Prit80].

With the method of block encryption, plaintext is broken into words and then encrypted on a word by word basis. The plaintext is transformed into ciphertext in segments of n words or blocks. For example, the federal Data Encryption Standard (DES) algorithm encrypts data in 64 bit blocks. The choice for which type of method to use is largely left to the application or the type of terminal equipment in use. Other factors which will effect the mode of encryption are the algorithm chosen, speed factors, cost, and the network protocol in use on the transmission link.

Conventional and public-key encryption are two algorithms that are currently in use as the foundation for authentication protocols. The basic difference between the two methods is the manner in which the keys are applied. Conventional methods can be characterized as single-key algorithms which leads to the problem of key distribution. It was the problem of how to distribute the keys securely that led to the development of public-key cryptosystems by Diffie and Hellman [Diff76].

2.5 Conventional Encryption

2.5.1 The Data Encryption Standard The best known of the modern techniques is the Data Encryption Standard, known as DES. It was officially adopted by the National Bureau of Standards in January 1977 as a federal standard for unclassified information [NBS77]. DES is a combination of substitution and transposition ciphers that encrypts a 64-bit plaintext input block into a 64-bit ciphertext output block. The algorithm utilizes a 64-bit key, of which 8 bits are for parity checking. The inverse process of decryption uses the same algorithm and makes use of the same 56 bit key.

The algorithm begins with a permutation of the 64 bits of plaintext. Then begins a series of 16 iterations of complex transformations. Each of these iterations is paramatorized by a function of the 56 bit key. The function consists of four steps, each of

which is carried out in sequence. The function forms the true complexity of the algorithm. It is a nonlinear, many-to-one substitution [Lemp79]. A typical iteration (Figure 3) uses 48 bits of the key, K_i , in a sequence which is described in the standard. The right most 32 bits, R_i , of the current permuted input is expanded to 48 bits by a linear operator, E_i . Then these bits are EXCLUSIVED-ORed with the key, K_i . The resulting 48 bits are divided into six-bit blocks, each of which is transformed by one of the 8 substitution boxes, S-boxes, into a four bit block. The resulting 32 bits are permuted and then EXCLUSIVED-ORed with the left most 32 bits, L_i . This 32 bit sum is joined with the 32 bit left side to form the current version of the 64 bit block.

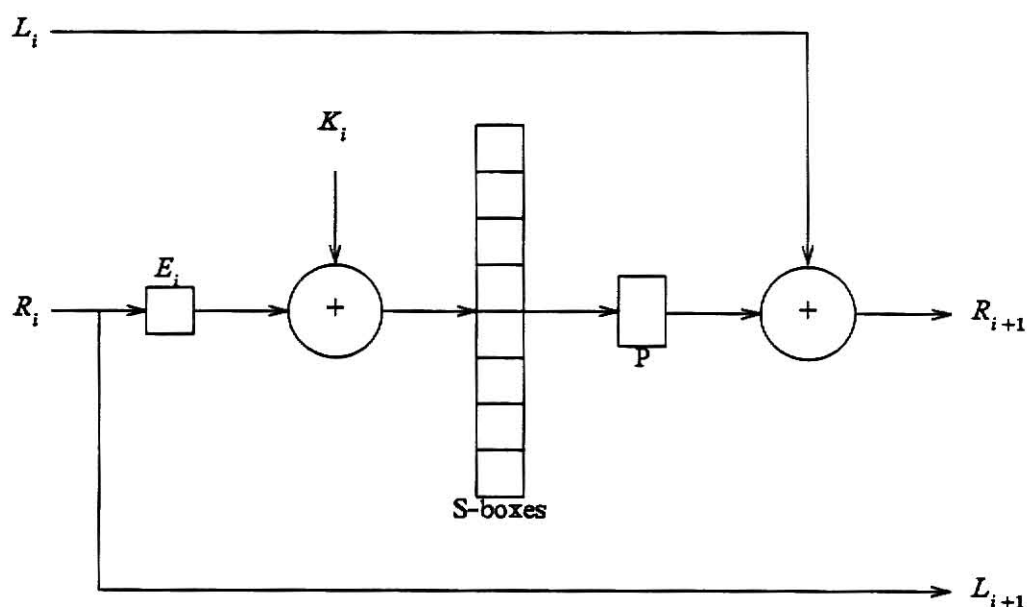


Figure 3. DES Iteration

The algorithm was designed for efficient hardware implementation, but it has also been implemented in software at the sacrifice of speed and throughput.

2.6 Public-key Encryption

Conventional methods are limited due to the difficulties involved with distributing the single key. Before communicants can establish a secure channel, they must wait until the secret key has been delivered by registered mail or trusted courier. To simplify this problem of key distribution, Diffie and Hellman devised a method in which communicating parties can communicate securely over the insecure channel without any prior communication [Diff76]. In this approach, each user wanting to send encrypted messages over the network would send out a public encryption key which would be published in an open directory. The user would retain a private decryption key which would be used to receive messages encoded with the public key. The underlying principle of this approach is that public disclosure of the encryption key does not compromise the secret decryption key.

Various implementations of this technique have been devised which are based on trapdoor one-way functions. A function is said to be one-way if it is an easily computed function, if it has an inverse function and if it is computationally infeasible to compute the inverse function. A trapdoor one-way function is a one-way function that has the additional property that the inverse function is easy to discover once specific information that was employed in the design of the function is known [Prit80]. Several public-key cryptosystems have been published most notably one by a group of computer scientists at Massachusetts Institute of Technology [Rive78] and another by Merkle and Hellman [Merk78].

2.7 Key Distribution

The most difficult requirement for implementing a secure channel with encryption is choosing the key to be used for the encryption algorithm and distributing it to both communicating parties. Due to the large number of users within a computer network, an

efficient means is necessary for distributing or managing the keys. Through the use of key distribution centers, KDC, the task of issuing keys can be removed from the individual hosts within the network [Kent78].

A KDC can be implemented on a secure host dedicated solely to the key distribution task. A KDC would hold two keys for each user that request its services, a primary key and a secondary key. All communication between the KDC and the host machine would be encrypted with the primary key. Messages between hosts would be encrypted with the secondary key.

This is but one possible implementation for key distribution protocol other methods could include multiple KDCs. The technique of multiple KDCs would involve a distributed database. Each KDC would have a replicated database consisting of primary and secondary keys.

2.8 Encryption Applications in Computer Networks

The use of encryption for providing security within a computer networks can provide a foundation for other network services. Authentication, network mail and digital signatures are applications that could be implemented across the secure channel.

2.8.1 Authentication Service The inherent principle of keys can be applied as a means for providing an authentication service. The authentication of messages, users and processes could be done across the network. Encryption can preserve the data integrity in a message exchange and conceal the destination and source addresses of messages sent through the network. It can also be used for applying access control to system resources. If a user must supply a password when logging into a system, encryption is used to protect the password while it is in storage. An application program could access a sensitive encrypted file only if the correct key is known.

2.8.2 Network Mail The same functions that an authentication service can provide could be extended to provide a reliable network mail facility. Typically an electronic mail service consists of a one-way communication path. The receiver may not be present when the mail is delivered to provide a two-way interactive conversation. This is due to the inherent delay between receiving the mail and reading the mail. But two-way authentication could be established through the use of encryption techniques.

2.8.3 Digital Signatures A person is bound to a contract or agreement by a handwritten signature. If a dispute arises, a signed contract can be presented in court as evidence of an agreement. To be able to replace the existing paper mail system with an electronic system, a method of providing a digital signature must be made possible. Digital signature techniques have been proposed that utilize both conventional and public-key algorithms.

2.9 A Survey of Authentication Protocols and Services

Distributed processing within computer networks, either local area networks or long-haul networks, has brought about a need for authentication protocols and services that provide a systematic approach to data security. Several protocols utilizing encryption techniques have been proposed and developed to provide a network authentication service.

2.9.1 Conventional and Public-key Approaches to Authentication Encryption techniques utilizing conventional and public-key methods have been shown to be a feasible means for establishing an authenticated connection, providing authenticated mail, signature verification and guaranteed message integrity [Need78]. Needham and Schroeder's approach to the authentication problem within computer networks was to provide a decentralized protocol with minimal reliance on network-wide services and naming authorities. Their method outlines a protocol that could be implemented using either conventional or public-key encryption algorithms. The concept of authentication servers and names servers is

introduced. An authentication server provides key management and interfaces to the name server database. An authentication server and a name server reside at each host.

A protocol utilizing conventional encryption could be implemented with the exchange of five (5) messages between users. A public-key algorithm implementation would require two additional messages. The extra messages result from the request for the public-keys. In both approaches the number of message exchanges could be reduced by utilizing local caching or storage of the keys. For the conventional approach, conversation keys could be stored for later reuse. Commonly used public-keys could be locally stored with the public-key implementation.

The implementation of the authentication servers is different for the two types of encryption chosen. In the conventional approach, the secret keys for each party must be encrypted to avoid browsing. But for the public-key method, the database consists of public-keys which do not need to be kept secret.

Needham and Schroeder have shown that authentication protocols utilizing conventional and public key means are very similar. The database for each approach is different due to the nature of their contents. They have shown that the public-key system has a noticeable advantage when implementing a digital signature service but can also be implemented using conventional techniques.

2.9.2 A Session Layer Authentication Service An authentication service designed within the session layer of a network architecture can provide a general model for a network environment [Chun84]. Chung and Sherman's design provides a means for insuring private communication between presentation entities. Their model is shown to provide secure session layer service despite the transport protocol utilized.

Their design consists of three components which are used to provide a name service, protection of session layer ports and to provide a connection mechanism. The name server provides four services:

- help: provides information on how to utilize the name server,
- registration: allows a user to enroll in the service,
- edit: allows a user to change his or her attributes within the name database,
- authentication: utilized by the authentication server for validating network entities.

The authentication server provides the connection and authentication for the communication between two processes. The encryption scheme used for all authentication messages between processes is the DES feedback cipher scheme. When a process requests a connection with another process, the authentication server is invoked to provide conversation keys to be used for encrypting the process to process messages.

2.9.3 An Office System Approach to Authentication An electronic mail facility, named Grapevine, was developed by the Xerox Corporation to provide a means for authentication and access control within a network of work stations and computers [Birr82]. Grapevine was implemented to provide computer mail service and provide research into the structure of distributed systems. The goals of their mail service were to provide authentication of senders and receivers, secure message delivery and database security.

2.9.4 A UNIX Secret Mail Facility A secret mail facility that runs under Berkeley UNIX 4.2 provides a secure communication channel between users on a local machine [BSD83]. This authentication service provides a means of sending and receiving encrypted mail. It provides commands that enable a user to enroll in the service and to send or receive messages. The service is integrated with ordinary UNIX mail. Ordinary mail

provides the announcement of secret mail. The encryption scheme which is used is a public-key cryptosystem. The key for the algorithm is established at enrollment and must be quoted in order to receive secret mail.

This service utilizes a directory of files which includes a key file for each user and message files. Ten message files may be contained in a user's mailbox at a time.

2.10 Comparisons and Contrasts

The design of the authenticated mail service, AMS, applies many of the techniques and principles of existing services. The scope of this paper deals with the authorization server. Similarities and differences can be seen when compared with some of the similar methods that provide the same form of authentication.

The Needham and Schroeder [Need78] approach for providing authentication in a computer network is very similar to the AMS approach. The methods that they conceived that were applied in the development of the AMS system include the following:

- the concept of decentralized servers,
- the notion of conversation keys between users,
- the implementation of multiple authentication servers,
- the concept of caching for local storage of keys,
- the implementation of one-way communication paths.

There are also some basic differences between the Needham and Schroeder approach and the AMS method which include the following:

- the task of creating the conversation key was assumed by the name server in the AMS system rather than the authentication server;

- there are no transaction stamps utilized with the AMS approach;
- AMS relies on timestamps for retrieval of conversation keys;
- AMS relies on "uucp" for handshaking between host machines and acknowledgement of messages;
- AMS provides double encryption between host machines;

The following similarities can be drawn when comparing AMS with Chung and Sherman's session layer design [Chun84]:

- both systems reside above the transport layer of a layered network;
- both systems apply access controls in the form of a password scheme;
- both systems were developed with the server concept;
- both systems utilize the DES algorithm.

Some of the differences between the two systems are the following:

- Chung and Sherman's design goal was to provide a session layer model for interfacing to different transport protocols. AMS design goal was to provide a secure presentation service,
- Unlike AMS, their prototype was designed using an Ada based design language,
- Unlike AMS, DES was used in feedback cipher scheme. AMS applies DES in electronic codebook form,
- AMS relies on the UNIX system command "uucp" for its line protocol.

AMS is very similar to the existing secret mail facility that Berkeley UNIX 4.2 provides.

The basic similarities are:

- both services reside in a UNIX environment;
- both services provide a secure channel via electronic mail;
- both services are implemented in C language;
- both services are integrated with ordinary mail;
- both services rely on a file structure for providing the registration database.

Upon closer examination there are many differences. Some of the differences include:

- AMS is both a network and local mail service, secret mail is only a local service,
- AMS utilizes a conventional encryption algorithm, secret mail utilizes a public-key cryptosystem,
- AMS was designed with the concept of servers,
- AMS has no effective limit upon the number of messages residing in a users mailbox; secret mail allows at most ten messages at a time.

From these comparisons, it can be concluded that most authentication services are very similar. Most of the differences are due to the various techniques used to solve the basic problem of encryption. Which is how to securely distribute the secret keys to the communicating parties.

3. Design and Implementation

3.1 Objective

The authenticated mail service, AMS, is a series of programs designed to provide a secure communication path between users on local or remote UNIX systems. It was designed to be simple for the user and to have a minimum amount of overhead. The system was developed in a UNIX programming environment and implemented in C language.

This chapter will discuss the Authorization Server module and the Message Server module. A detailed description of the Name Server is the subject of a report to be published [Merr86]. The Name Server module consists of functions that perform various tasks for the Authorization Server and the Message Server. These tasks involve the registration database, which is a file system consisting of user information, machine information and key information. This chapter will describe the design of the system, the operation of each program, the installation of the system and administration of the system. Manual pages in Appendix 1 contain detailed syntactic and semantic descriptions of the system's commands.

A term used throughout this chapter is "UUCP network" [Nowi82]; this is a network of UNIX based machines interconnected by the public telephone network.

3.2 Design Assumptions

The following design assumptions were made in the development phase of the authenticated mail service:

1. To provide authenticated mail to those that choose to communicate securely.
2. A user's machine is a member of the UUCP network.

3. The users commands should have similar syntax to existing commands that utilize the UUCP network.
4. The user is responsible for the security of the message after it is securely delivered through the network.
5. An ordinary mail service is available for delivering authenticated mail notification.
6. A single notice concerning arrival of authenticated mail is sufficient for notification.

3.3 System Description

The system consists of three modules (Figure 4). The modules have been dubbed servers.

They are:

1. Authorization Server (AS) - provides the user interface to the service.
2. Name Server (NS) - manages the registration database.
3. Message Server (MS) - provides the mail delivery mechanism and manages the mailboxes.

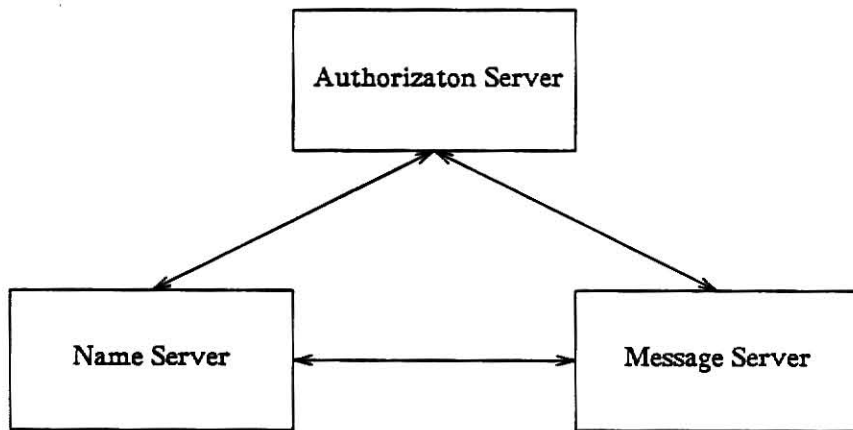


Figure 4. Modules of the Authenticated Mail Service

3.4 User Commands

The AS consists of four primary and two secondary programs for interfacing to the service:

- Enrollx
- Connectx
- Sendx
- Disconnectx
- Rsendx
- Rconx

"Rsendx" and "rconx" are secondary programs that are executed by "uux" on remote machines. They are the system's interface to the UUCP network. The "uux" command is a communication facility of the network.

The MS consists of one primary program for delivering mail and managing the user's mailbox, namely,

— Getx.

3.4.1 Enrollx The 'enrollx' command allows a user to enroll in the service and establish a password so that secret messages can be sent. This command is the user's initial interface to the system.

After entering the enrollx command, the user is prompted for a password which will subsequently have to be quoted each time the service is accessed. The user is requested to reenter the password to detect any mistakes that might have occurred during the initial password entry. If the two don't agree, the user is requested to enter a new password. The program calls an NS function to register the user's logname and password in the registration database. A message is printed to the user after successful completion of the enroll command. This message acknowledges enrollment in the service.

3.4.2 Connectx The 'connectx' command allows a user to establish a connection with a second user on a local or remote machine. The connection is established in the form of a conversation key (CK). To denote a recipient on a remote system, a person's name is prefixed by the system name and an exclamation point. After the user has entered the command, the program prompts for the user's password. If the password is incorrect the program halts execution.

The conversation key established with the connect command is used for encrypting messages in both directions of communication. This key is recorded as part of the information stored by the Name Servers associated with the communicating parties.

If a remote connection is being established, the conversation key is encrypted using the destination machine's secret AS key. For example, if user A on machine A establishes a connection with user B on machine B, the ensuing message sent to machine B would be of the form:

$$AS_A \rightarrow AS_B : \{ A, CK \}^{K_{AS_B}}$$

The notation introduced here which is derived from Needham and Schroeder [Need78] will be followed through out this chapter: AS_A is authorization server on A machine, AS_B is authorization server on B machine, K_{AS_B} is the secret key of the authorization server on B machine and encryption is indicated by braces followed by the key used. The key (for the conversation between A and B) and A's name is encrypted with B's authorization server's secret key. The encrypted message is then sent to the remote machine via the 'uux' command.

3.4.3 Sendx This command allows a user to send an encrypted message to a second user enrolled in the network wide service. Initially, the user is prompted for the password which was established at enrollment.

After quoting the correct password, the program is in the message collection mode. A message prompting the user to enter the message is printed to the user. "Sendx" will gather up the message until a line consisting of just a period and a newline is entered. The recipient of the message is notified by ordinary mail [Kern84] that authenticated mail has arrived. Prior to delivery, the message is encrypted with the conversation key which was previously established with a "connectx" command. If the command is a remote send, the message is doubly encrypted with the destination party's authorization server's secret key and sent via the "uux" command. The message would be of the following form:

$$AS_A \rightarrow AS_B : \left\{ A : B : \{ Message \}^{CK_{AB}} \right\}^{K_{AS_B}}$$

After receipt of the above message the AS of B machine would decrypt it and place the following message in B's "secret" mailbox:

$$(A:B:\{Message\}^{CK_{AB}})$$

3.4.4 Disconnectx The purpose of this command is to allow a user to remove a previously established connection either remotely or locally. A user should not issue a disconnect without acknowledgements from the UUCP network regarding any outstanding remote connects. This helps to maintain the integrity of the registration database. After the user has correctly entered his password, the program will wipe out any trace of a connection between the user issuing the command and the target user. If there are any unread messages in a user's secret mailbox, the key will not be destroyed until they are read. But no new messages can be sent with the key.

3.4.5 Getx This command implements the message server. After the user has correctly entered the password, his or her system mailbox is scanned for messages. If there are messages to be read, a cleartext header is read to determine the sender's address. A name server function is then invoked to return the conversation key. After the name server returns the conversation key, the message is decrypted and presented to the user. The messages are read using a last in first out (LIFO) order of presentation to the user. For each message, the user is prompted with a '?', and a line is read from standard input [Kern84] to determine the disposition of the message. The following options are available:

- | | |
|-----------|--|
| <newline> | Put message back in mailbox and go on to next message. |
| d | Delete message and go on to next message. |
| s [files] | Save the message in the named files. |
| q | Put mail back in the mailbox and stop. |
| !command | Escape to the shell [Kern84] to execute command. |
| ? | Give a summary of these options. |

If unread messages are returned to a user's mailbox, they are placed in their encrypted

form. If new mail arrives while a user is reading his messages, a notification of new mail is given.

3.5 Detailed Description

This section provides a detailed description of the system programs with emphasis on data flow through each process. During the execution of the system, one process is running initially. Each of the programs spawn child processes [Kern84] for performing various tasks. Some of these tasks include:

- notification of authenticated mail.
- performing remote execution via the UUCP network.
- execution of shell commands.

Many of the routines used within the programs consist of system calls and library subroutines [UNIX84]. The standard I/O library [Kern84] is used extensively through out the primary and secondary programs that comprise the system.

3.5.1 Files and Data Types Various files and data types are created and removed during runtime execution. Each user enrolled in the service has a file representing his secret mailbox. The mailfile is located in the authenticated mail directory. A mailfile is created when a secret message addressed to the user arrives. The message is appended to this file. If a mailfile already exists, the message is appended to the end of the file. The user mailfile is removed when all the messages have been read. During execution of the programs, temporary and lock files are created and removed to preserve file integrity during I/O operations.

The object type "structure" [Kern78] is the primary link between the authorization server, message server and the name server. All communication with the NS is performed

via the structure. The structure is initialized by the AS and the MS and passed to the NS. The name server accesses the user information file and loads up the members of the structure with user attributes (Figure 5). Throughout, the name "user structure" will be used when referring to this structure.

```
static struct DATABASE {
    int FLAG;           /* valid or invalid request */
    int CMD;            /* type of request */
    char *usr_id;        /* user's logname */
    char usr_pswd[25];   /* user's password */
    char *mach_id;       /* remote machine name */
    char *rem_usr_id;    /* remote user's id */
    char c_key[12];      /* conversation key */
    char as_key[25];     /* authorization server key */
}
```

Figure 5. User Data Structure

The DES initialization tables [NBS77] are represented by static character arrays. These tables are utilized by the DES encryption function and are available through a system subroutine.

3.5.2 Signals The primary vehicle for interprocess communication is signals [Kern84]. Signals are sent to notify a process that a certain event has occurred. The process determines what happens when a signal is received.

Various signals are caught throughout the system programs that comprise the authenticated mail service. This enables the programs to exit gracefully. For example, if an interrupt, hangup or quit signal is received, the program will close any open files and remove any lock files that have been created.

3.5.3 Enrollx Description As described previously, 'enrollx' is the command that allows a user to enroll in the authenticated mail service. A detailed description of the program is now given.

To enroll in the authenticated mail service, a user simply enters the command name (see Appendix 1). The program first determines the logname of the user executing the command; it initially tries the "getlogin" subroutine to perform the task. If the subroutine returns a NULL pointer indicating the name was not found, the program checks the system password file [Kern84] for the name entry. All users who have logins on the machine will have an entry in the password file containing their logname and various information. If an error or an end of file (EOF) occurs before the user's logname is found, the program prints an error message indicating the error and stops execution. Once the user's logname is determined, the program generates a key to be used for encrypting the user's password. The encryption method chosen is the same scheme used with the system login procedure [Morr79]. The scheme is a one-way function that employs the DES algorithm.

When a password is first entered, the program generates a 12-bit random number. The random number is generated by reading the real-time clock and summing the result with a process id. The summation is used to perturb the DES algorithm in 1 of 4096 ways. One of the internal tables of the DES algorithm, the so-called E-table [NBS77], is changed based on the 12-bit number. After a password is typed in, the program prompts for the password just entered in. A comparison is made and if they don't match, the user is asked to re-enter a new password. This is a mechanism to catch any typing mistakes that might have occurred while entering a new password.

The password is used as a key for encrypting a constant string consisting of zero's. The encrypted result is then passed to the Name Server (NS) to be entered in the user's registration database. The NS returns a flag indicating valid or invalid enrollment. If an invalid flag is returned, an error message is printed. Otherwise, a message indicating enrollment is sent to the user. The NS will update the files on the local and remote machines.

The password chosen will subsequently be prompted for when a user attempts to use an AMS command. The following procedure is done to compare the password that is typed in and the one stored in the user's enroll file. The 12-bit quantity is stripped off the stored password and is used to perturb the DES algorithm in the same manner. The password is then used as a key to encrypt the same constant string. If the comparison fails, a message is printed indicating the failure and the program halts.

3.5.4 Connectx and Rconx Description The "connectx" command can be syntactically used in two ways (see Appendix 1). The user can either enter the person's logname on the command line or be prompted. After entering the correct password, the program checks for the number of arguments on the command line. If there is more than one, the second argument is taken as the person's logname. A prompt is sent to the user if only one argument is found on the command line. The prompt asks for the address. The address can either consist of a logname or both a logname and a machine name. A local connect only requires the user's logname. The program makes a function call to determine what the input string consisted of, i.e., whether a local or remote connection.

The "getem" function is passed a string consisting of the target address. The function examines the string a character at a time. If an exclamation mark is found, the characters scanned up to the mark are defined as the machine name. The next string of characters up to a newline character is taken as the user's logname. The user's logname and machine name are loaded into global variables defined by the main routine of the "connectx" command. If a local connect is being requested, the local machine name is loaded into the machine name variable. After completion, the "getem" function will return one of three values. A zero is returned if the connect was local, a one if the connect was remote or a minus one if the string contained an error.

Upon successful return, the members of the user structure are loaded with the variables set by the "getem" function. Otherwise, an error message is printed to the "stderr" file [Kern84]. The member of the structure that indicates to the NS the type of command being executed is set, and the address of the structure is passed to the NS. The NS checks the integrity of the data passed, i.e., valid machine names and valid user names, and proceeds to load the structure with all the information required. If no errors are found, the NS will generate a conversation key in the form of a nine digit integer. The database is updated with the information and a valid flag is set and control is returned to the main routine.

If no errors are found, the program will either spawn a process to perform a remote connection or exit successfully. The remote connection requires that the program create an encrypted message consisting of the originating user's logname and the newly created conversation key. The key used for encrypting the message is the destination machine's AS secret key. This key is entered into an available machines file by the system administrator as a turn-up procedure. The NS loads up the user structure with the AS key on the initial call.

The basic interprocess communication facility used to perform remote execution is the "pipe" [Kern84]. A pipe is a special type of file designed for efficient transfer of data between processes. A pipe is opened between the "connectx" command and the "uux" command. The program writes the encrypted message to the pipe, and the uux process reads from the pipe. The "popen" system subroutine performs all synchronization and overhead between the two processes. Once the message is completely written to the pipe, the main routine of the connect process checks for any errors that might have occurred with the pipe communication. If none occurred, the program exits successfully.

To complete the remote connection, the "uux" command opens a pipe between itself and

the "rcon" program residing on the remote machine. The "rcon" program is initiated by the "uux" command. The encrypted message is written to the pipe by "uux" and "rconx" reads it from the pipe. Initially, the main routine of "rcon" will call it's local NS to return the AS key necessary to decrypt the message. After decryption, the members of the structure are loaded with the information received from the remote machine and the structure is passed to the NS. The NS performs data checking for validity and sets a flag indicating the updated result.

If the update was successful an exit status of zero is returned; otherwise, a minus one is returned. The exit status is used as an acknowledgement to the originating machine that the remote connection was successful. A UUCP network function is to return the exit status of the remotely executed program. The status is returned via remote mail to the originator of the "connectx" command. The user can view the exit status of the "rconx" program as an acknowledgement of the remote connect.

3.5.5 Sendx and Rsendx Description These commands implement the message gathering and delivery mechanism. The command line consists of the command name and the receiver's address (see Appendix 1). The program begins by determining the sender's logname and machine name. The password is prompted for and is compared to the password returned by the NS. If the comparison fails, the program stops; otherwise, execution continues. A temporary file is created which is used for gathering the input message. If an error occurs while attempting to open the file for writing, the program prints a message to "stderr" and exits. The program reads the real time clock and converts it into a format to be used in the header of the message. The sender's name and machine name are also included in the header string. The second line of the header block is the destination logname.

The second argument of the command line is then passed to the "getem" function to determine the destination address. After "getem" returns, the header string is generated with the sender and time information. If the command is a remote send, additional information will be on the header line. This information is used by "rsendx" for decrypting the message after delivery through the network. Once the header line is built, the NS is invoked to return various information. This information will consist of the conversation key and the AS key if the command is a remote send. If it is a remote send, the header is encrypted and written to the temporary file. Otherwise, the cleartext header is written to the file.

Once the header has been written, the program enters the message gathering mode. A call is made to the "getmsg" function. This function gathers the message from standard input and writes it to the temporary file until a line consisting of just a period followed by a newline is entered. In addition, the "getmsg" function calls the "des" function for encrypting the message with the conversation key returned by the NS. The message is encrypted on a per line basis. Double encryption using the destination machine AS key is performed on remote messages. After the input message is read and encrypted, the temporary file is written to the receiver's mailfile. The function "sendlocal" is called for a local message. For remote messages, the function "sendrmt" is called. These functions perform the temporary to mailfile copy routine.

The "sendlocal" function generates a lock file that will disable the mailfile. Other processes will not be able to access the file during the copying process. A call from within the function is made to a copy routine. This routine appends the encrypted message at the end of the user's mailfile. Error checking is done at various points with appropriate messages printed if any occur.

The "sendrmt" function performs an identical task in addition to interfacing to the UUCP network. Instead of appending the message to the mailfile, a pipe is opened in the same manner as the "connectx" command. The message is written into the pipe, and the "uux" command reads from the pipe. The "uux" command creates another pipe to the "rsendx" program on the remote machine. The "rsendx" program is very similar to the "sendx" program except the doubly encrypted message read from the pipe is decrypted once with its AS secret key. The singly encrypted message is then written to the user mailfile. Error checking is also performed in this function.

The program then generates a notification message for the receiver. A notification is generated only with a local send. The "rsendx" program creates a notification for a remote send. The task of sending the notice is performed by a child process. A child process is generated by a "fork" system call [Kern84]. The child process is then overlayed with a copy of the ordinary mail program using it's standard input. A notification message which is declared at initialization of the main program is written to the mail process's standard input. The main routine or parent process [Kern84] waits for the child process to complete before continuing with execution. Once the mail program or child process has completed, the program flushes it's buffers, closes open files, and removes any work files.

3.5.6 Disconnectx Description This command enables a user to wipe out any trace of a connection with a second user. The target address can either be entered on the command line as an argument or it can be entered when prompted (see Appendix 1). As with all the commands, the user is prompted for a password. After the password has been verified, the program retrieves the target address string. A call is made to the "getem" function to derive the address information and determine if it is a remote or local disconnection. A call is then made to the NS. The structure is passed to the NS with the user's logname, second user's logname and machine name and the type of command being executed. The task of

performing a remote disconnect is left to the NS due to the disabling of the conversation keys within the registration database.

3.5.7 Getx Description The "getx" command implements the Message Server. It delivers the messages and manages the user mailfiles. To check a secret mailbox, a user simply enters the "getx" command (see Appendix 1). After successfully determining the user's logname, the program prompts for the password. The program checks the authenticated mail directory to determine if the user has any mail. If a mailfile does not exist, then a message is printed to the user, and the program exits.

If there is mail to be read, the program creates a temporary file and copies the contents of the user's mailfile into it. The messages are transferred to the temporary file in their encrypted form. The mailfile is locked out while the transfer is in process to prevent access to the file by another process. If there are multiple messages in the mailfile, an addressing scheme is used to determine the starting and stopping points of messages in the file. The scheme utilizes a structure with an integer address as a member. As the messages are read into the temporary file, each line is examined for a "From:" marker that signifies the start of a message. When a marker is found, the line number is copied into the address structure to be used for reading the messages out. The mailfile lock is removed and the messages are printed out in LIFO order.

The messages are decrypted by the "decipher" function. Initially, the header line is examined to determine whom the message is from. The sender's name, the message timestamp and the machine name are loaded into the user structure and passed to the NS. After successful return of the conversation key from the NS, lines are read from the temporary file. As each encrypted line is read, they are passed to the encryption function "des" with the conversation key. The "des" function returns the cleartext which is printed

out to the user. At the end of each message, a question mark prompting the user for a response is printed out. A valid response is an "s" and a file name, which indicates that the message should be saved in the given file. A file will be created in the user's home directory [Kern84], and the message will be appended. Another valid response is either a "d" or newline. If the response is a "d", the message which was just read will be deleted and the next message if any will be printed out. A "newline" causes the message which was just read to be put back in the user's secret mailbox and the next message will be printed out. If there is no more mail to be read the program will quit at that point. A third response is the letter "q". This instructs the program to quit and leave any unread messages in the mailfile. To get a summary of options available, a user can enter a "?". After all messages have been read or the user quits the program, the temporary file is removed, and any unread messages are put back into the mailfile. The mailfile lock is removed, and the program exits normally.

3.5.8 The Encryption Subroutine The "des" subroutine implements the Federal DES standard [NBS77]. The routine was built using a variation of the password encryption algorithm employed on the UNIX system [Morr79]. That function consists of three routines: crypt, setkey and encrypt. Crypt is the password function and setkey and encrypt provide access to the actual DES algorithm including the perturbation tables. The crypt subroutine was modified to obtain the desired function while the setkey and encrypt routines were simply used as subroutine calls. The arguments to the function are text, a key and a crypt flag. To encrypt text, the function is sent blocks of eight bytes and a flag set to zero instructing the function to perform encryption. The plaintext is converted to its binary form and passed to an "encrypt" entry within the function. After the "encrypt" entry returns, the 64-bit block of ciphertext is transformed into 11 six bit words. The words are converted into printable ascii characters in the set (a-zA-Z0-9./) by adding or subtracting a constant from

each word.

To decrypt text, the function is passed multiples of eleven bit blocks and the key. The ciphertext is converted to its original character set and then passed to the "encrypt" entry which is run in reverse order to generate the plaintext. The plaintext is passed back to the calling function.

3.6 User Scenario

As a summary to the previous descriptions, a user scenario will be given. The scenario will depict a possible user session and the steps involved in sending and receiving authenticated mail. Figure 6 shows the message exchange that would take place. The arrows in the figure indicate the flow of the messages that are being sent or received.

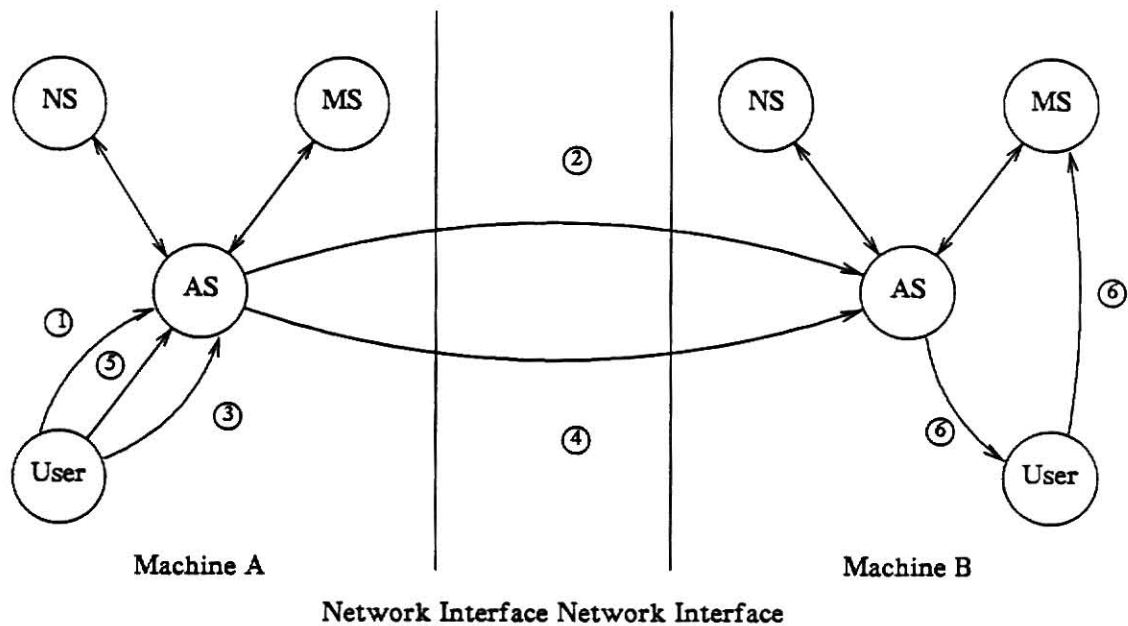


Figure 6. Message Exchange in a Typical User Session

Assume that user A on machine A and user B on machine B have previously enrolled in the service. If user A is sending a message to user B, the following exchange would take place:

1. User A would first issue a "connectx" command requesting a connection with User B on machine B.
2. The "connectx" command would send the conversation key to User B's AS in ciphertext. User B's AS would then decrypt it and call its NS to enter the key in the registration database.
3. After receiving acknowledgement via remote mail that the connection has been established, User A issues a "sendx" command addressed to User B.
4. The "sendx" command gathers the message and sends it to User B's machine via the UUCP network. The "rsendx" program places the message in User B's secret mailbox and sends a notification of authenticated mail to User B.
5. After receiving acknowledgement from remote mail that the message had been delivered to User B, User A can now issue a "disconnectx" command wiping out any trace of a connection.
6. After the notification of authenticated mail has been received, User B issues a "getx" command to receive the authenticated mail. The unencrypted message can be saved in a file with the permission mode set by User B.

3.7 System Administration

To administer the Authenticated Mail Service, it is recommended that a directory named "smail" be created for this purpose. Subdirectories for the source code, user commands bin and mailboxes should be created within the smail directory. A system administrator would execute a makefile [Kern84] in the source code directory to bring the system up. The usr

commands would automatically be placed in the bin directory, and user secret mailfiles would be created under the mailbox directory. The system administrator would perform some Name Server functions such as loading up the Authorization server keys for the various machines included in the service. For the users to access the system, a path pointing to the smail directory should be added to their ".profile".

4. Conclusion and Summary

The Authenticated Mail Service provides a mechanism for users to communicate securely across a publicly accessible medium. The design is tailored for the UUCP network and relies on many existing facilities.

4.1 Design Choices

When the service was in its development stage, two approaches were considered. One was a datagram approach that implemented a one-way communication path. A second approach was a handshaking, two-way communication path. The first approach was chosen due to the nature of an electronic mail service. Because of the inherent delay between receiving mail and reading mail, a one-way path was chosen.

Two choices were available in integrating the UUCP network into the service. There was a choice between using the "uux" or the "uucp" commands. The "uux" command was chosen because it came the closest to emulating real time execution on a remote machine. There are numerous problems associated with the "uucp" command. Some of which include the following:

- the files reside in a spool directory with very little protection,
- no remote processing is performed,
- it is only a file transfer facility.

The choice for which command to use was a rather easy one. Using "uux" enabled a number of things to be done. The execution of remote commands allowed the following:

- the decryption of files,
- the setting of permission modes,

— the placing of files into target directories.

4.2 Future Enhancements

As a possible enhancement to the service, a two-way virtual circuit could be added as a feature. It would require that a host machine be able to communicate with another host machine in real time with minimum delay between exchange of messages. This author feels the user would have to expect some overhead in the form of delay. It appears to be feasible if another network service other than UUCP is considered.

Another enhancement to the service would be to provide a more sophisticated message server. More options can be provided to the user. An option such as allowing a user to forward messages to other users, rather than saving them in a file and then forwarding them, is one possible extension.

As mentioned previously, key distribution is the most difficult task of implementing a cryptosystem. One approach that could be used for the key distribution would be to integrate a public-key cryptosystem into the service. The public-key method could be used for distributing the conversation keys. In the present service, they are distributed by conventional methods using the authorization server's secret keys. By implementing a public-key system, the preliminary step of delivering the authorization server's secret key to each machine would be eliminated.

4.3 Conclusion

The Authenticated Mail Service was a fairly extensive project. There are still a number of questions to be answered concerning its robustness in light of failures during remote transfers. Questions remain regarding the possibility of junk files due to destroyed or lost keys. But those are administration problems. The service as it exists in its prototype stage has met its design intent, namely, to provide a secure communication channel.

REFERENCES

- [Birr82] Birrell, A. D., Levin, R., Needham, R. M., Schroeder, M. D., "Grapevine: An Exercise in Distributed Computing", Communications of the ACM, Vol. 25, No. 4, April 1982.
- [Chun84] Chung, A., Sherman, R., "The Design of an Authentication Service", Computer Communication Review, Vol. 13-14, No. 5, 1, January - April 1984.
- [Davi84] Davies, D. W., Price, W. L., "Security for Computer Networks", Wiley, New York, 1984.
- [Denn82] Denning, D. E., "Cryptography and Data Security", Addison-Wesley, Reading, Mass., 1982.
- [Diff76] Diffie, W., and Hellman, M. E., "New Directions in Cryptography", IEEE Transactions in Information Theory, Vol. IT-22, November 1976a.
- [Gram84] Grampp, F. T., Morris, R. H., "Unix Operating System Security", Bell System Technical Journal, Vol. 63, No. 8, Part 2, October 1984.
- [Isra83] Israel, J. E., Linden, T. A., "Authentication in Office System Internetworks", ACM Transactions on Office Information Systems, Vol. 1, No.3, July 1983.
- [Kent81] Kent, S. T., "Security in Computer Networks", in "Protocols and Techniques for Data Communication Networks", Kuo, F. F. (ed.), Prentice-Hall, New Jersey, 1981.
- [Kern84] Kernighan, B. K., Pike, R., "The UNIX Programming Environment", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.
- [Kern78] Kernighan, B.K., Ritchie, D. M., "The C Programming Language", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.
- [Kitt83] Kittelberger, K. L., "Scope of Computer Security Problems", in "Advances in Computer Security Management", Wofsey, M. W. (ed.), Wiley, New York, 1983.
- [Lemp79] Lempel, A., "Cryptology in Transition", Computing Surveys, Vol. 11, December 1979.
- [Merk78] Merkle, R. C., "Secure Communications Over Insecure Channels", Communications of the ACM, Vol. 21, April 1978.
- [Merr86] Merritt, J. W., "Distributed File Systems in an Authentication System", Master's Report, Kansas State University, 1986.
- [Morr79] Morris, R., Thompson, K., "Password Security: A Case History", Communications of the ACM, Vol.22, November 1979.
- [NBS77] National Bureau of Standards, "Data Encryption Standard", Federal Information Processing Standards, Publication 46, January 1977.
- [Need78] Needham, R. M., Schroeder, M. D., "Using Encryption for Authentication in Large Networks of Computers", Communications of the ACM, Vol. 21, No. 12 December 1978.

- [Nowi82] Nowitz, D.A., Lesk, M.E., "Implementation of a Unix Network", Computer Communications, Vol. 5, February 1982.
- [Pope79] Popek, G. J. and Kline, C. S., "Encryption and Secure Computer Networks", Computing Surveys, Vol. 11, December 1979.
- [Rive78] Rivest, R. L., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public-key Cryptosystems", Communications of the ACM, Vol. 21, February 1978.
- [Simm79] Simmons, G. J., "Symmetric and Asymmetric Encryption", Computing Surveys, Vol. 11, December 1979.
- [Tann81] Tannenbaum, A. S., "Computer Networks", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.
- [UNIX84] "UNIX System V - Release 2.0 Programmer Reference Manual", AT&T Technologies, April 1984.
- [BSD83] "UNIX Programmer Manual", 4.2 Berkeley Software Distribution, Virtual VAX-11 Version, University of California, Berkeley, CA, August 1983.
- [Wood81] Wood, C. C., "Future Applications of Cryptography", Proc. of the 1981 Symposium on Security and Privacy, IEEE, 1981.

Appendix 1

User Command Manual Pages

enrollx ()

UNIX Programmer's Manual (local)

enrollx ()

NAME

enrollx - enroll in the authenticated mail service

SYNOPSIS

enrollx

DESCRIPTION

Enrollx allows a person to enroll in the authenticated mail service.

The program prompts for a password that will subsequently have to be quoted each time a command of the authenticated mail service is invoked. The password is prompted for twice. Only the first eight characters of the password are significant.

A password may be changed by invoking the "editx" command of the authenticated mail service.

FILES

/usr/local/smali/USERS
/usr/local/smali/MACHINES
/usr/local/smali/ERRLOG

SEE ALSO

helpx, sendx, getx, connectx, disconnectx, editx

connectx ()

UNIX Programmer's Manual (local)

connectx ()

NAME

connectx - establish a connection with another user within the authenticated mail service

SYNOPSIS

connectx
connectx person
connectx system!person

DESCRIPTION

Connectx establishes a connection with another user either remotely or locally who is enrolled in the authenticated mail service. The connection is established in the form of a conversation key which is used for encrypting and decrypting messages. Authenticated mail can be sent only after a connection has been established.

The program initially prompts for a password that was established at enrollment into the service.

FILES

/usr/local/smail/mail/* secret mailboxes

SEE ALSO

enrollx, getx, sendx, disconnectx, editx

sendx ()

UNIX Programmer's Manual (local)

sendx ()

NAME

sendx - send authenticated mail

SYNOPSIS

sendx person
sendx system!person

DESCRIPTION

Sendx allows a person to send authenticated mail either remotely or locally within the authenticated mail service. The command is like mail(1), but no one but the indented recipient can read the messages. The program utilizes conventional encryption methods for implementing a secure communication channel. Before authenticated mail can be sent, a connection must be established using the "connectx" command.

To send a message to another user, *sendx* is invoked with the person who is being sent the message as an argument. The command will accept only one person as a recipient. To send remote mail, the argument to the program is the system name followed by an exclamation point and the user's name.

The program initially prompts for a password that was established at enrollment into the service.

The program accepts the message until a line with just a period and a newline is entered.

FILES

/usr/smail/mail/* secret mailboxes

SEE ALSO

enrollx, getx, connectx, disconnectx, editx

getx ()

UNIX Programmer's Manual (local)

getx ()

NAME

getx - receive authenticated mail

SYNOPSIS

getx

DESCRIPTION

Getx allows a user to receive authenticated mail. Mail is presented in last-in-first-out order (LIFO). After each message is printed, the user is prompted with a question mark. The user must tell the program how to dispose of the message that was just printed. The following options are available:

<newline>	put message back in mailbox and go on to the next message
d	delete the message and go on to the next message
s filename	save the message in the named file
!command	escape to the shell and perform the command
q	quit the program
?	print a summary of these options

The program initially prompts for a password that was established at enrollment into the service.

FILES

/usr/local/smail/mail/* secret mailboxes

SEE ALSO

enrollx, disconnect, sendx, connect

disconnectx ()

UNIX Programmer's Manual (local)

disconnectx ()

NAME

disconnectx - remove a previously established connection

SYNOPSIS

disconnectx

disconnectx person

disconnectx system!person

DESCRIPTION

Disconnectx removes a connection that was established with another user either remotely or locally using the "connectx" command. The program wipes out any trace of a connection between users. After a disconnect, authenticated mail can no longer be sent to the target of the disconnection. Another connection must be established.

The program initially prompts for a password that was established at enrollment into the service.

FILES

/usr/local/smail/mail/* secret mailboxes

SEE ALSO

enrollx, getx, sendx, connectx, editx, helpx

editx ()

UNIX Programmers Manual (local)

editx ()

NAME

editx - change an authenticated mail service password.

SYNOPSIS

editx

DESCRIPTION

Editx allows users to change their password for accessing the authenticated mail service.

The command prompts users for their old password. It then prompts for the new password twice. Only the first eight characters of the password are significant. The new password will subsequently be entered when executing an authenticated mail command.

FILES

/usr/local/smail/USERS
/usr/local/smail/MACHINES
/usr/local/smail/ERRLOG

SEE ALSO

helpx, sendx, getx, connectx, disconnectx

helpx ()

UNIX Programmer's Manual (local)

helpx ()

NAME

helpx - provide help information on the authenticated mail service

SYNOPSIS

helpx

DESCRIPTION

Helpx is used to provide information on the commands of the authenticated mail service. It provides general information on the system and prints the system command manual pages.

Helpx examines the environment variable **\$TERM** to select options that adapt the output to the terminal being used.

FILES

/usr/local/smail/USERS

/usr/local/smail/MACHINES

/usr/local/smail/ERRLOG

SEE ALSO

enrollx, sendx, getx, connectx, disconnectx

Appendix 2

Application Code

```
/* connect.c    6/16/86
 *
 * usage: conn [user | sys!user]
 *
 * This cmd establishes a conversation key between
 * users on local or remote systems. If it is
 * a remote connection "uux" is used to send the key
 * over the network. This module calls the Name Server
 * to create the key and enter it into the registration file.
 * Also, if needed the AS key is returned if it is a remote
 * connection. The cmd accepts as input an addressee in the
 * form of a machine id (if connection is remote) and a user
 * name. If the addressee is not given on the cmd line the
 * user is prompted for that info. Upon successful completion,
 * a value of 0 is returned indicating the connection was
 * established. Otherwise, a value greater than 1 is returned
 * indicating an error occurred.
 */
```

```
#include "DATA.BASE"
```

```
#define TRUE 1
```

```
#define FALSE 0
```

```
struct passwd *getpwuid();
char who[20];
char sys[20];
char localsys[20];
char *getlogin();
char *getpass();
char *ckey;
char *akey;
char *des();
int getem();
```

```
main(argc, argv)
```

```
char **argv;
```

```
int argc;
```

```
{
    FILE *rmf, *popen();
    char cmd[200];
    register char *namep;
    char *crypt();
    char *myname;
    char name[20];
    char line[50];
    char *lp;
    char *pw;
    char *pwd, opwbuf[25];
    char *salt;
```

```
register i;
int n, uid, c, rmt;
struct DATABASE *p_DB;

for (i = SIGHUP; i < SIGTERM; i++)
    signal(i, SIG_IGN);
p_DB = &DB;
uid = getuid();
myname = getlogin();
if(myname == NULL)
    myname = getpwuid(uid)->pw_name;
p_DB->cmd = PASSWORD;
p_DB->usr_id = myname;
NS_Request(p_DB);
if (p_DB->FLAG == INVALID) {
    fprintf(stderr, "connect: database access error\n");
    exit(1);
}
strcpy(opwbuf, getpass("password: "));
pwd = p_DB->usr_pswd;
pw = crypt(opwbuf, pwd);
if(strcmp(pw, pwd) != 0) {
    fprintf(stderr, "Sorry.\n");
    exit(1);
}

strncpy (name, "", sizeof(name));
if (argc > 1)
    strncpy(name, argv[1], sizeof(name));
while (name[0] == '\0') {
    namep = name;
    printf("addressee: ");
    while ((c = getchar()) != '\n') {
        if (c == EOF)
            exit(0);
        if (namep < name + sizeof(name))
            *namep++ = c;
    }
}
rmt = getem(name); /* get addressee info and set remote flg */
p_DB->rem_usr_id = whoto;
p_DB->mach_id = sys;
p_DB->cmd = CONNECT;
#ifdef DEBUG
printf("whoto = %s\n", whoto);
printf("sys = %s\n", sys);
#endif
NS_Request(p_DB);
if(p_DB->FLAG == INVALID) {
    fprintf(stderr, "connect: data base access error\n");
```

```
        exit(2);
    }
    ckey = p_DB->C_key;
#ifdef DEBUG
    printf("ckey = %s\n", ckey);
#endif
    akey = p_DB->AS_key;
#ifdef DEBUG
    printf("akey = %s\n", akey);
#endif
    if(rmt) {
        sprintf(line, "%s %s %s %s", myname, localsys, whoto, ckey);
        lp = des(akey, line, 0);
#ifdef DEBUG
        printf("encrypted line = %s\n", lp);
#endif
        /*
         * Use uux to issue remote connect
         */
        sprintf(cmd, "/usr/bin/uux - %s!rcon", sys);
#ifdef DEBUG
        printf("uux cmd = %s\n", cmd);
#endif
        if ((rmf=popen(cmd, "w")) == NULL) {
            fprintf(stderr, "connect: error opening the pipe\n");
            exit(2);
        }
        n = strlen(lp);
        if (write(rmf->_file, lp, n) != n) {
            fprintf(stderr, "connect: cannot pipe to rcon command\n");
            perror(write);
            exit(1);
        }
        /*
         * check status
         */
        if (pclose(rmf) != 0) {
            fprintf(stderr, "connect: error closing the pipe\n");
            exit(1);
        }
    }
    exit(0);
}
```

```
/* disconnect.c      7/8/85 */
*
* usage : disconnect [ user| systuser ]
*
* This command removes a conversation key that
* was previously established between users on a local or
* remote system. If the disconnect is a remote
* disconnection, the "uux" utility is used to
* send the disconnection to the remote system.
*
*/

#include "DATA.BASE"

struct passwd *getpwuid();
char who[20];
char sys[20];
char *getlogin();
char *getpass();
int getem();

main(argc, argv)
char **argv;
int argc;
{
    char cmd[200];
    register char *namep;
    char *crypt();
    char *myname;
    char name[20];
    char *pw;
    char *pwd, opwbuf[25];
    int uid, c;
    struct DATABASE *p_DB;
    register i;

    for (i = SIGHUP; i < SIGTERM; i++)
        signal(i, SIG_IGN);
    p_DB = &DB;
    uid = getuid();
    myname = getlogin();
    if (myname == NULL)
        myname = getpwuid(uid)->pw_name;
    p_DB->cmd = PASSWORD;
    p_DB->usr_id = myname;
    NS_Request(p_DB);
    if (p_DB->FLAG == INVALID) {
        fprintf(stderr, "disconnect: database access error\n");
        exit(1);
    }
}
```

```
    }
    strcpy(opwbuf, getpass("password: "));
    pwd = p_DB->usr_pswd;
    pw = crypt(opwbuf, pwd);
    if(strcmp(pw, pwd) != 0) {
        fprintf(stderr, "Sorry.\n");
        exit(1);
    }

    strncpy(name, "", sizeof(name));
    if (argc > 1)
        strncpy(name, argv[1], sizeof(name));
    while (name[0] == ' ') {
        namep = name;
        printf("addressee: ");
        while ((c = getchar()) != '\n') {
            if (c == EOF)
                exit(0);
            if (namep < name + sizeof(name))
                *namep++ = c;
        }
    }
    getem(name); /* get addressee info */
    p_DB->rem_usr_id = whoto;
    p_DB->mach_id = sys;
    p_DB->cmd = DISCONNECT;
#ifdef DEBUG
    printf("whoto = %s\n", whoto);
    printf("sys = %s\n", sys);
#endif
    NS_Request(p_DB);
    if(p_DB->FLAG == INVALID) {
        fprintf(stderr, "disconnect: database access error\n");
        exit(2);
    }
    exit(0);
}
```

```
/*      enroll.c              6/16/86
*
* This command allows a user to enroll in the authenticated
* mail service (smail). It requires a login id and a passwd
* that will have to be quoted each time an smail service cmd
* is invoked. The input info is passed to the Name Server
* which creates the entry in the registration database.
*
*/

#include "DATA.BASE" /* user registration database structure */

int getuid();
int quit();
char *getpass();
char *getlogin();
char *crypt();
struct passwd *getpwuid();
extern int errno;

main()
{
    struct DATABASE *p_DB;
    char saltc[2];
    char *mypasswd;
    char pwbuf[10], npwbuf[10];
    long salt;
    int i, c, uid;
    int tries = 0;
    int loop;

    for (i = SIGHUP; i < SIGTERM; i++)
        setsig(i, quit);
    p_DB = &DB;
    p_DB->usr_id = getlogin();
    if(p_DB->usr_id == NULL) {
        uid = getuid();
        p_DB->usr_id = getpwuid(uid)->pw_name;
    }

    /*
    * The following code generates a random number
    * by adding the long integer time with a process
    * id. The number is "processed" and used as the key for
    * the perturbation of the "DES" algorithm within the "crypt"
    * subroutine call which is used to "hide" the user passwd
    *
    */

    time(&salt);
```

```
salt += getpid();
saltc[0] = salt & 077;
saltc[1] = (salt >> 6) & 077;
for (i=0; i<2; i++) {
    c = saltc[i] + '.';
    if (c>'9') c += 7; if (c>'Z') c += 6;
    saltc[i] = c;
}
fprintf(stderr, "Enter a password to be used for accessing the secret
mail service\n");
loop = 1;
while (loop) {
    strcpy(pwbuf, getpass("password: "));
    strcpy(npwbuf, getpass("Re-enter password: "));
    if (strcmp(pwbuf, npwbuf)){
        if (++tries > 2) {
            fprintf(stderr, "Too many tries; try again later.\n");
            exit(1);
        }
        else
            fprintf(stderr, "They don't match; try again.\n");
    }
    else
        loop = 0;
}
mypasswd = crypt(pwbuf, saltc);
strcpy(p_DB->usr_pswd, mypasswd);

/*
 * The Name Server is called to enter the user in the
 * registration database.
 */

for ( i = SIGHUP; i < SIGTERM; i++)
    signal(i, SIG_IGN);
p_DB->cmd = ENROLL;
p_DB = NS_Request(p_DB);
if(p_DB->FLAG == VALID ) {
    printf("You are now enrolled in the smail service\n");
    exit(0);
}
else {
    fprintf(stderr, "enroll: registration database error.\n");
    exit(1);
}
}
setsig(i, f)
int i;
int (*f)();
{
```

```
        signal(i, f);
    }
    quit()
    {
        perror(errno);
        exit(1);
    }
```

```
/*      getem.c      6/16/86
*
* This function is called to return a destination
* address when an addressee is passed as an
* argument. It loads up external variables with
* an addressee's name and system. If the addressee
* is on a remote system a value of 1 is returned,
* otherwise a value of 0 is returned indicating a
* local address.
*/
#define KSUVAX
#include <stdio.h>

extern char whoto[];
extern char sys[];
extern char localsys[];

int getem(name)
char *name;
{
    char *namepp, *p;
    int rmt = 0;
#ifdef KSUVAX
    gethostname(localsys, sizeof(localsys));
#endif
#ifdef KSU832
    uucpname(localsys);
#endif
/*
* check for 'l' indicating a remote address
*/
    for(p = name; *p != 'l' && *p != ' '; p++)
        ;
    namepp = name;
    if(*p == 'l')
    {
        rmt = 1;
        for(p = sys; *namepp != 'l'; *p++ = *namepp++)
            ;
        *p = ' ';
        namepp++;
        for(p = whoto; *namepp != ' '; *p++ = *namepp++)
            ;
        *p = ' ';
    }
    else
    {
        strcpy(sys, localsys);
        strcpy(whoto, name);
    }
}
```

```
return(rmt);  
}
```

```
/*      getx.c      6/16/86
*
* usage: getx
*
* Getx allows a user to receive authenticated mail.
* Mail is presented in LIFO order. After each message
* the user is prompted with a question mark to
* determine what to do with the mail.
*
*/
#include "DATA.BASE" /* registration database */

#define BAD          -1
#define TRUE         1
#define FALSE        0
#define ERROR        -1
#define LOCKON       1
#define LOCKOFF      0
#define MODE         700
#define LSIZE        BUFSIZ
#define TEXT         243
#define READ         4
#define ERRFILE      2
#define ERRSPACE     3
#define MAXLET       300

FILE *tmpf, *smalf;

/* structure containing pointers into letter file */
struct let {
    long adr;
    char change;
} let[300];

struct stat stbuf;
struct passwd *getpwent(), *getpwuid();
char smaildir[] = SMAILDIR;
char smailtmp[] = TMPFILE;
char smaillock[] = ".lock";
char from[] = "From ";
char to[] = "To: ";
char *opt[] = {
    "newline      put message back in mailbox and go on to the next message\n",
    "d            delete the message and go on to the next message\n",
    "s filename    save message in named file\n",
    "!command     escape to the shell and perform the command\n",
    "q            quit the program\n",
    "?"          print a summary of theses options\n",
    0};

char curlock[50];
char line[LSIZE];
char resp[LSIZE];
```

```
char savfile[50];
char *myname;
char *home;
char smailfile[50];
char *getlogin();
char *getenv();
char *des();
char *crypt();
extern int errno;
int nlet;
int error;
int locked;
int done();
int changed;
struct DATABASE *p_DB;
main()
{
    char *pw;
    char *pwd, opwbuf[25];
    int stret, aret;
    int print, i, j;
    int quit;
    setbuf(stdout, malloc(BUFSIZ));
    if(((home = getenv("HOME")) == NULL) || (strlen(home) == 0))
        home = ".";
    myname = getlogin();
    if((myname == NULL) || (strlen(myname) == 0))
        myname = getpwuid(geteuid())->pw_name;
    for (i = SIGHUP; i <= SIGTERM; i++)
        signal(i, SIG_IGN);
    p_DB = &DB;
    p_DB->cmd = PASSWORD;
    p_DB->usr_id = myname;
    NS_Request(p_DB);
    if (p_DB->FLAG == INVALID) {
        fprintf(stderr, "getx: database access error\n");
        exit(1);
    }
    strcpy(opwbuf, getpass("password: "));
    pwd = p_DB->usr_pswd;
    pw = crypt(opwbuf, pwd);
    if(strcmp(pw, pwd) != 0) {
        fprintf(stderr, "Sorry.\n");
        exit(1);
    }
    mktemp(smailtmp);
    unlink(smailtmp);
    if ((tmpf = fopen(smailtmp, "w")) == NULL) {
        fprintf(stderr, "getx: can't open %s for writing\n", smailtmp);
        exit(2);
    }
}
```

```
scat(smailfile, smaildir, myname);
stret = stat(smailfile, &stbuf);
if((aret = access(smailfile, READ)) == NULL)
    smalf = fopen(smailfile, "r");
if (stret == NULL && aret == BAD) {
    fprintf(stderr, "getx: permission denied!\n");
    exit(2);
}
else
    if (smalf == NULL || stbuf.st_size == NULL) {
        printf("No authenticated mail.\n");
        unlink(smailtmp);
        exit(2);
    }
lock(smailfile);
copytmp(smalf, tmpf);
fclose(smalf);
fclose(tmpf);
unlock();
if((tmpf = fopen(smailtmp, "r")) == NULL) {
    fprintf(stderr, "getx: can't open %s\n", smailtmp);
    exit(2);
}
changed = 0;
quit = 0;
print = 1;
setsig(SIGINT, done);
setsig(SIGQUIT, done);
setsig(SIGHUP, done);
for (i = 0; i < nlet; ) {
    j = nlet - i - 1;
    if (print)
        decipher(j, stdout);
    printf("? ");
    fflush(stdout);
    if(fgets(resp, sizeof(resp), stdin) == NULL)
        break;
    switch(resp[0]) {
        case 'n':
        case '\n':
            i++;
            break;
        case 'd':
            let[j].change = 'd';
            changed++;
            i++;
            break;
        case '!':
            system(resp+1);
            printf("!\n");
            print = 0;
    }
}
```

```
        break;
    case 'q':
        quit=1;
        break;
    case 's':
        if (resp[1] != ' '){
            printf("invalid command, no file name\n");
            print=0;
            continue;
        }
        umask(7);
        sscanf(resp, "s %s\n", fname);
        if (t = access(fname, 0) == NULL)
            printf("mail being appended to '%s'\n", fname);
        smalf = fopen(fname, "a");
        rewind(tmpf);
        decipher(j, smalf);
        fclose(fname);
        let[j].change = 'd';
        changed++;
        i++;
        break;
    case '?':
        {
            register i;
            for(i=0; opt[i]; i++)
                printf("%s", opt[i]);
        }
        print=0;
        break;
    }
    if (quit)
        break;
}
signal(SIGINT, SIG_IGN);
signal(SIGQUIT, SIG_IGN);
signal(SIGHUP, SIG_IGN);
if (changed)
    putback();
fclose(tmpf);
unlock();
unlink(smailtmp);
exit(0);
}

/*
 * copy mail to temp directory in encrypted form
 */
copytmp(f1, f2)
register FILE *f1, *f2;
{
```

```
char *p, *pp;
char buf[BUFSIZ];
long nextadr;
int n;
int i, nl=0;
int toflag = 0;

nlet = nextadr = 0;
let[0].adr = 0;
while (fgets(line, sizeof(line), f1) != NULL) {

    if(line[0] == from[0])
        if (isfrom(line))
            let[nlet++].adr = nextadr;
    n = strlen(line);
    nextadr += n;
    if (write(f2->_file, line, n) != n) {
        fclose(f1);
        fclose(f2);
        fprintf(stderr, "getx: no space for temp file\n");
        exit(2);
    }
}

/*
 * last plus 1
 */
let[nlet].adr = nextadr;
}

/*
 * transforms ciphertext to plaintext and copies to stdout
 *   li      -> index into letter structure
 *   f       -> copy to file
 */
decipher(li, f)
register FILE *f;
{
    register int i, ii;
    register char *s, *p;
    char buf[TEXT], lastc, ch;
    char tmpbuf[TEXT];
    char timestamp[25], sys[25];
    char *header;
    char sender[50];
    int n, nn, j;
    long k;
    int num;
    int len;
    int toflag=0, nl=0;

    fseek(tmpf, let[li].adr, 0);
```

```
k = let[li+1].adr - let[li].adr;
for(i=0;i<k;) {
    if (fgets(buf, sizeof(buf), tmpf) == NULL)
        exit(2);
    nn = strlen(buf);
    if(buf[0] == from[0])
        if (isfrom(buf)) {
            sscanf(buf, "From %s ", sender);
            header = index(buf, '(');

#ifdef DEBUG
printf("header = %s\n", header);
#endif

            len = nn - (strlen(header) + 1);
            sscanf(header, "( %s %s )", sys, timestamp);
            strncpy(tmpbuf, buf, len);

#ifdef DEBUG
printf("%s\n", tmpbuf);
#endif

            strcpy(buf, tmpbuf);
            strcat(buf, "\n");

#ifdef DEBUG
printf("%s\n", buf);
printf("sender = %s\n", sender);
printf("sys = %s\n", sys);
printf("timestamp = %s\n", timestamp);
#endif

            p_DB->rem_usr_id = sender;
            p_DB->mach_id = sys;
            strcpy(p_DB->C_key, timestamp);
            p_DB->cmd = MSERVER;
            NS_Request(p_DB);
            if (p_DB->FLAG == INVALID) {
                fprintf(stderr, "getx: database access error\n");
                unlink(smailtmp);
                exit(2);
            }

#ifdef DEBUG
printf("ckey = %s\n", p_DB->C_key);
#endif

            toflag = 0;
        }
    if (toflag > 1) {
        if (index(buf, '\n')) {
            ii = strlen(buf);
            buf[ii-1] = NULL;
            nl = 1;
        }
        p = des(p_DB->C_key, buf, 1);
        strcpy(buf, p);
        if (nl)
            strcat(buf, "\n");
    }
}
```

```
    }
    n = strlen(buf);
    if (write(f->_file, buf, n) != n)
        exit(2);
    toflag++;
    i += nn;
    fflush(f);
}
return(TRUE);
}

/*
 * get letter
 *   li    -> index into letter file
 *   f     -> copy to file
 */
getlet(li, f)
register FILE *f;
{
    register int i;
    register char *s;
    char    buf[BUFSIZ], lastc, ch;
    int     n, j;
    long    k;
    int     num;

    fseek(tmpf, let[li].adr, 0);
    k = let[li+1].adr - let[li].adr;
    if(k)
        k--;
    for(i=0; i<k; i++) {
        s = buf;
        num = ((k-i) > sizeof(buf)) ? sizeof(buf) : (k-i);
        if((n = read(tmpf->_file, buf, num)) <= 0)
            return(FALSE);
        lastc = buf[n-1];
        if(write(f->_file, buf, n) == ERROR)
            return(FALSE);
        i += n;
        fflush(f);
    }
    if( lastc != EOF) {
        read(tmpf->_file, buf, 1);
        write(f->_file, buf, 1);
    }
    return(TRUE);
}

/*
 * put temp messages back in mailboxes
 */
putback()
```

```
{
    register i, n, c;
    int new = 0, aret;
    struct stat stbuf;

    lock(smailfile);
    stat(smailfile, &stbuf);
    /*
     * Check to see if new messages arrived
     */
    if (stbuf.st_size != let[nlet].adr) {
        if((smalf = fopen(smailfile, "r")) == NULL) {
            fprintf(stderr, "getx: can't re-read %s\n", smailfile);
            error = ERRFILE;
            done();
        }
        fseek(smalf, let[nlet].adr, 0);
        fclose(tmpf);
        if((tmpf = fopen(smailtmp, "a")) == NULL) {
            fprintf(stderr, "getx: can't re-write %s\n", smailfile);
            error = ERRFILE;
            done();
        }

        /*
         * append one and only one new message
         */
        while ((c = fgetc(smalf)) != EOF)
            if (fputc(c, tmpf) == EOF) {
                fclose(smalf);
                fclose(tmpf);
                fprintf(stderr, "getx: no space for tmp file\n");
                error = ERRSPACE;
                done();
            }
        fclose(smalf);
        fclose(tmpf);
        if((tmpf = fopen(smailtmp, "r")) == NULL) {
            fprintf(stderr, "getx: can't re-read %s\n", smailtmp);
            error = ERRFILE;
            done();
        }
        if(nlet == (MAXLET-2)) {
            fprintf(stderr, "putback: Too many letters\n");
            error = ERRSPACE;
            done();
        }
        let[++nlet].adr = stbuf.st_size;
        new = 1;
    }
    /*
```

```
* Put messages back in mailbox
*/
if((aret=access(smailfile, 2)) == 0)
    smalf = fopen(smailfile, "w");
if ((smalf == NULL) || (aret == ERROR)) {
    fprintf(stderr, "getx: can't rewrite %s\n", smailfile);
    error = ERRFILE;
    done();
}
rewind(tmpf);
n = 0;
for (i = 0; i < nlet; i++)
    if (let[i].change != 'd') {
        if (getlet(i, smalf) == FALSE) {
            fprintf(stderr, "getx: cannot copy mail back\n");
            perror(errno);
            exit(2);
        }
        n++;
    }
fclose(smalf);
/*
 * If all letters are read, call Message Server
 * to remove deactivated keys, then remove
 * user's smailfile
 */
if (n == 0) {
    p_DB->cmd = MSERVER;
    p_DB->FLAG = DELETE;
    NS_Request(p_DB);
    unlink(smailfile);
}
if (new)
    printf("new mail arrived\n");
unlock();
}
/*
 * check for "from" pattern in header string
 */
isfrom(lp)
register char *lp;

{
    register char *p;

    for (p = from; *p; )
        if (*lp++ != *p++)
            return(FALSE);
    return(TRUE);
}
/*
```

```
* check for "to" string pattern in header string
*/
isto(lp)
register char *lp;

{
    register char *p;

    for (p = to; *p; )
        if (*lp++ != *p++)
            return(FALSE);
    return(TRUE);
}
/*
* create smail lock file with 700 mode
*/
lock(file)
char *file;
{
    register int f, i;

    if (locked == LOCKON)
        return;
    scat(curlock, file, smaillock);
    for (i=0; i<10; i++) {
        if((f = creat(curlock, MODE)) != ERROR){
            close(f);
            locked = LOCKON;
            return;
        }
        sleep(2);
    }
    unlock();
    unlink(smailtmp);
    perror(errno);
    exit(2);
}

unlock()
{
    unlink(curlock);
    locked = LOCKOFF;
}
/*
* concatenate from1 and from2 to to
*   to      -> destination string
*   from1   -> source string
*   from2   -> source string
* return:
*   none
*/
```

```
scat(to, from1, from2)
register char *to, *from1, *from2;
{
    for (; *from1;)
        *to++ = *from1++;
    for (; *from2;)
        *to++ = *from2++;
    *to = ' ';
}
/*
 * strlen - determine length of line (terminated by '\n')
 */

strlen(s)
char *s;
{
    int i;

    for (i=0 ; i < LSIZE && s[i] != '\n' ; i++);
    return(i+1);
}

setsig(i, f)
int i;
int (*f)();
{
    signal(i, f);
}

/*
 * clean up and exit gracefully
 */
done()
{
    unlock();
    unlink(smailtmp);
    exit(error);
}
```

```
/* rcon.c      6/16/86
 *
 * Rcon is the command executed by remote
 * systems via the uux command. It decrypts
 * the connect info and passes it to the
 * name server.
 */

#include "DATA.BASE"

#define DEBUG
#define TRUE      1
#define FALSE     0

#ifdef DEBUG
char debug[] = "/usrb/att/chance/smail/USERS/debug";
#endif

main()
{

#ifdef DEBUG
FILE *fp;
#endif

    struct DATABASE *p_DB;
    char whoto[50];
    char sys[50];
    char myname[50];
    char ckey[50];
    char line[256];
    char *lp;
    char *key;
    char *akey;
    char *des();
    char *p;

    scanf("%s", line);
#ifdef DEBUG
    fp = fopen(debug, "w");
    fprintf(fp, "encrypted line = %s\n", line);
#endif
    p_DB = &DB;
    p_DB->FLAG = ASKEY;
    p_DB->cmd = CONNECT;
    NS_Request(p_DB);
    if (p_DB->FLAG == INVALID)
        exit(2);
    lp = des(p_DB->AS_key, line, 1);
    sscanf(lp, "%s %s %s %s", whoto, sys, myname, ckey);
```

```
#ifdef DEBUG
fprintf(fp, "AS_key = %s\n", p_DB->AS_key);
fprintf(fp, "whoto = %s\n", whoto);
fprintf(fp, "sys = %s\n", sys);
fprintf(fp, "myname = %s\n", myname);
fprintf(fp, "ckey = %s\n", ckey);
#endif

strncpy(p_DB->C_key, ckey, strlen(ckey));

#ifdef DEBUG
fprintf(fp, "C_key = %s\n", p_DB->C_key);
fclose(fp);
#endif
    p_DB->usr_id = myname;
    p_DB->rem_usr_id = whoto;
    p_DB->mach_id = sys;
    p_DB->FLAG = REMOTE;
    p_DB->cmd = CONNECT;
    NS_Request(p_DB);
    if(p_DB->FLAG == INVALID)
        exit(2);
    exit(0);
}
```

```
/*      rsendx.c  6/16/86
 *
 * Rsendx is the command executed by
 * remote systems via the uux command.
 * It decrypts the messages and passes
 * the message to the message server.
 *
 */

#include "DATA.BASE"
#define DEBUG
#define LSIZE      256
#define BUFSIZE    512
#define TEXT       243
#define TRUE       1
#define FALSE      0
#define ERROR      -1

extern int errno;
struct let {
    long adr;
    char change;
} let[300];

FILE *tmpf, *smalf;

char smailtmp[] = TMPFILE;
char smaildir[] = SMAILDIR;
char nospace[] = "sendx: no space for tmp file\n";
char buf[128];
char sys[20], whoto[20];
char localsys[20];
char *akey;
char *des();
char *p;

#ifdef DEBUG
char debug[] = "/usrb/att/chance/smail/mail/debug";
FILE *fp;
#endif

int getem();
int nlet = 0;
int ret;
struct DATABASE *p_DB;

main(argc, argv)
char **argv;
{
    char file[50];
```

```
    int rmt, n=0;

#ifdef DEBUG
fp = fopen(debug, "w");
#endif

    mktemp(smailtmp);
    unlink(smailtmp);
    tmpf = fopen(smailtmp, "w");
#ifdef DEBUG
    if (tmpf == NULL) {
        fprintf(fp, "sendx: cannot open %s for writing\n", smailtmp);
        exit(1);
    }
#endif
    rmt = getem(++argv);
    p_DB = &DB;
    p_DB->FLAG = ASKEY;
    p_DB->cmd = CONNECT;
    NS_Request(p_DB);
    if (p_DB->FLAG == INVALID) {
        unlink(smailtmp);
        exit(1);
    }

#ifdef DEBUG
fprintf(fp, "askey = %s\n", p_DB->AS_key);
fclose(fp);
#endif

    akey = p_DB->AS_key;
    if (getmsg() == FALSE) {
        fclose(tmpf);
        unlink(smailtmp);
        fprintf(stderr, nospace);
        exit(1);
    }
    nlet = 1;
    let[0].adr = 0;
    let[1].adr = ftell(tmpf);
    if (fclose(tmpf) == EOF) {
        fclose(tmpf);
        unlink(smailtmp);
        fprintf(stderr, nospace);
        exit(1);
    }
    if ((tmpf = fopen(smailtmp, "r")) == NULL) {
#ifdef DEBUG
        fprintf(fp, "rsendx: cannot reopen %s for reading\n", smailtmp);
#endif
        exit(1);
    }
}
```

```
    }
    scat(file, smaildir, whoto);
    smalf = fopen(file, "a");
    if (smalf == NULL) {
#ifdef DEBUG
        fprintf(fp, "rsendx: cannot append to %s\n", file);
#endif
        unlink(smailtmp);
        exit(1);
    }
    if (getlet(0, smalf) == FALSE) {
#ifdef DEBUG
        fprintf(fp, "rsendx: cannot append to %s\n", file);
#endif
        unlink(smailtmp);
        exit(1);
    }
    fclose(smalf);
    fflush(tmpf);
    fclose(tmpf);
    if (ret = unlink(smailtmp) < 0)
        exit(1);
    exit(0);
}

/*
 * get message from stdin, call
 * encryption routine and put in
 * tmp file
 */

getmsg()
{
    char line[LSIZE], *p;
    char buf[BUFSIZE];
    int cord=0;
    int i, len;
    while (fgets(line, TEXT, stdin) != NULL) {
        if (line[0] == '.' && line[1] == '\n')
            break;
        p = des(akey, line, 1);
        if (fputs(p, tmpf) == EOF)
            return(FALSE);
    }
    return(TRUE);
}

/*
 * concatenate from1 and from2 to to
 * to -> destination string
 * from1 -> source string
 */
```

```
*      from2 -> source string
* return:
*      none
*/
scat(to, from1, from2)
register char *to, *from1, *from2;
{
    for (; *from1;)
        *to++ = *from1++;
    for (; *from2;)
        *to++ = *from2++;
    *to = ' ';
}
/*
* get letter
*      li      -> index letter table
*      f       -> copy to file
*/
getlet(li, f)
register FILE *f;
{
    register int i;
    register char *s;
    char    buf[BUFSIZE], lastc, ch;
    int     n, j;
    long    k;
    int     num;

    fseek(tmpf, let[li].adr, 0);
    k = let[li+1].adr - let[li].adr;
    if(k)
        k--;
    for(i=0; i<k; i++) {
        s = buf;
        num = ((k-i) > sizeof(buf)) ? sizeof(buf) : (k-i);
        if((n = read(tmpf->_file, buf, num)) <= 0)
            return(FALSE);
        lastc = buf[n-1];
        if(write(f->_file, buf, n) == ERROR)
            return(FALSE);
        i += n;
        fflush(f);
    }
    if( lastc != EOF) {
        read(tmpf->_file, buf, 1);
        write(f->_file, buf, 1);
    }
    return(TRUE);
}
```

```
/*      sendx.c      6/16/86
*
* usage: sendx person | sys!person
*
* Sendx allows a person to send authenticated mail
* either remotely or locally within the authenticated
* mail service. To send a message to another user,
* the command is invoked with the person who is being
* sent the message as an argument.
*
*/

#include "DATA.BASE" /* user registration database */

#define KSUVAX
#ifdef KSUVAX
#include "/usr/wb/uucp/uucp.h"
#endif

#define LSIZE      256
#define BUFSIZE    512
/* amount of text encryption algorithm requires */
#define TEXT      257
#define RMTMSG    " remote from %s"
#define TRUE      1
#define FALSE     0
#define ERROR     -1
#define LOCKON    1
#define LOCKOFF   0
#define MODE      700

/* structure containing pointers into letter file */
struct let {
    long adr;
    char change;
} let[300];

FILE *tmpf, *malf;

extern int errno;

char *timezone();
char sys[20], who[20], localsys[20];
char smailtmp[] = TMPFILE;
char smaildir[] = MAILDIR;
char smaillock[] = ".lock";
char nospace[] = "sendx: no space for tmp file\n";
char notice[] = "you have authenticated mail";
char curlock[50];
char buf[256];
char *ckey;
```

```
char *akey;
char *des();
char *myname;
char *thissys;
char *getenv();
char *getlogin();
char *crypt();
char *p;

long iop;
int getem();
int done();
int locked;
int nlet = 0;
int ret;
int rmt;

main(argc, argv)
char **argv;
{
    struct DATABASE *p_DB;
    struct tm *bp, *localtime();
#ifdef KSUVAX
    struct timezone tz;
    struct timeval tv;
#endif
    char file[50];
    char pp[128];
    char *pw;
    char *pwd, opwbuf[25];
    char *zp, *tp;
    char hdr[20];
    char timestamp[20];
    int n=0;
    register i;

    if ( argc != 2 ) {
        fprintf(stderr, "sendx: send to exactly one person\n");
        exit(1);
    }
#ifdef KSU832
    uucpname(thissys);
#endif
#ifdef KSUVAX
    gethostname(thissys, sizeof(thissys));
#endif
    if (((myname = getenv("LOGNAME")) == NULL) || (strlen(myname) == 0))
        myname = getlogin();
    if ((myname == NULL) || (strlen(myname) == 0))
        myname = getpwuid(geteuid())->pw_name;
    for (i = SIGHUP; i < SIGTERM; i++)
```

```
        signal(i, SIG_IGN);

/*
 * Load up user structure for Name Server
 */

p_DB = &DB;
p_DB->usr_id = myname;
p_DB->cmd = PASSWORD;
NS_Request(p_DB);
if (p_DB->FLAG == INVALID) {
    fprintf(stderr, "sendx: database access error");
    exit(1);
}
strcpy(opwbuf, getpass("password: "));
pwd = p_DB->usr_pswd;
pw = crypt(opwbuf, pwd);
if(strcmp(pw, pwd) != 0) {
    fprintf(stderr, "Sorry.\n");
    exit(1);
}
mktemp(smailtmp);
unlink(smailtmp);
tmpf = fopen(smailtmp, "w");
if (tmpf == NULL) {
    fprintf(stderr, "sendx: cannot open %s for writing\n", smailtmp);
    exit(1);
}
time(&iop);
sprintf(timestamp, "%ld", iop);
bp = localtime(&iop);
tp = asctime(bp);
#ifdef KSUVAX
    gettimeofday(&tv, &tz);
#endif
zp = timezone(bp->tm_min, bp->tm_isdst);
rmt = getem(++argv);
if (rmt) {
    sprintf(buf, "From %s %.16s %.3s %.4s", myname, tp, zp, tp+20);
    if (ret = sprintf(pp, RMTMSG, thissys) < 0) {
        printf(stderr, "return value was %d", ret);
        exit(1);
    }
    strcat(buf, pp);
}
else
    sprintf(buf, "From %s %.16s %.3s %.4s ", myname, tp, zp, tp+20);
sprintf(hdr, "( %s %s )\n", localsys, timestamp);
strcat(buf, hdr);
p_DB->cmd = SEND;
p_DB->rem_usr_id = whoto;
```

```
p_DB->mach_id = sys;
NS_Request(p_DB);
if(p_DB->FLAG == INVALID) {
    fprintf(stderr, "sendx: database access error\n");
    unlink(smailtmp);
    exit(2);
}
akey = p_DB->AS_key;
ckey = p_DB->C_key;
#ifdef DEBUG
printf("ckey = %s\n", ckey);
printf("askey = %s\n", akey);
#endif
    if (rmt) {
        p = des(akey, buf, 0);
        fputs(p, tmpf);
    }
    else
        fputs(buf, tmpf);
    sprintf(buf, "To: %s\n", whoto);
    if (rmt) {
        p = des(akey, buf, 0);
        fputs(p, tmpf);
    }
    else
        fputs(buf, tmpf);
    setsig(SIGHUP, done);
    setsig(SIGINT, done);
    setsig(SIGQUIT, done);
    if (getmsg() == FALSE) {
        fclose(tmpf);
        fprintf(stderr, nospace);
        exit(1);
    }
    signal(SIGHUP, SIG_IGN);
    signal(SIGINT, SIG_IGN);
    signal(SIGQUIT, SIG_IGN);
    nlet = 1;
    let[0].adr = 0;
    let[1].adr = ftell(tmpf);
    if (fclose(tmpf) == EOF) {
        fclose(tmpf);
        fprintf(stderr, nospace);
        exit(1);
    }
    if((tmpf = fopen(smailtmp, "r")) == NULL) {
        fprintf(stderr, "sendx: cannot reopen %s for reading\n", smailtmp);
        exit(1);
    }
    if (rmt) {
        if ( sendrmt( sys, whoto) == FALSE)
```

```
        printf(stderr, "sendx: cannot send to specified user\n");
    }
    else {
        if ( sendlocal(whoto) == FALSE )
            printf(stderr, "sendx: cannot send to specified user\n");
    }
    fflush(tmpf);
    fclose(tmpf);
    if ( ret = unlink(smailtmp) < 0 )
        perror(errno);
    /* generate notification using ordinary mail */
    sprintf(buf, "mail %s <%snotice", whoto, smaildir);
    if (system(buf) < 0)
        perror("smail");
    exit(1);
}

/*
 * get message from stdin, call
 * encryption routine and put in
 * tmp file
 */

getmsg()
{
    char line[LSIZE], *p;
    char buf[BUFSIZE];
    int cord=0;
    int i, len;
    fprintf(stderr, "Enter message:\n");
    while (fgets(line, TEXT, stdin) != NULL) {
        if (line[0] == '.' && line[1] == '\n')
            break;
        p = des(ckey, line, cord);
        if (rmt) {
            strcpy(buf, p);
            p = des(akey, buf, cord);
        }
        if (fputs(p, tmpf) == EOF)
            return(FALSE);
    }
    if (rmt) {
        p = des(akey, "\n", cord);
        if (fputs(p, tmpf) == EOF)
            return(FALSE);
    }
    else
        if (fputs("\n", tmpf) == EOF)
            return(FALSE);
    return(TRUE);
}
```

```
sendlocal(whoto)
char whoto[20];
{
    char file[50];
    FILE *malf;
    scat(file, smaildir, whoto);
    lock(file);
    malf = fopen(file, "a");
    if (malf == NULL) {
        fprintf(stderr, "sendx: cannot append to %s\n", file);
        exit(1);
    }
    if (getlet(0, malf) == FALSE) {
        fprintf(stderr, "sendx: cannot append to %s\n", file);
        exit(1);
    }
    fclose(malf);
    unlock();
    return(TRUE);
}

sendrmt( rsys, name )
char *rsys;
char *name;
{
    FILE *rmf, *popen();
    char cmd[200];

    /*
     * Use uux to send secret mail
     */

    sprintf(cmd, "/usr/bin/uux - %s!rsend %s", rsys, name);
#ifdef DEBUG
    printf("%s\n", cmd);
#endif
    /*
     * write message to a pipe
     */

    if ((rmf=popen(cmd, "w")) == NULL)
        return(FALSE);
    if (getlet(0, rmf) == FALSE) {
        fprintf(stderr, "sendx: cannot pipe to mail command\n");
        pclose(rmf);
        return(FALSE);
    }
    /*

```

```

    * check status
    */
    return(pclose(rmf)==0 ? TRUE : FALSE);
}
/*
 * concatenate from1 and from2 to to
 *   to      -> destination string
 *   from1   -> source string
 *   from2   -> source string
 * return:
 *   none
 */
scat(to, from1, from2)
register char *to, *from1, *from2;
{
    for (; *from1;)
        *to++ = *from1++;
    for (; *from2;)
        *to++ = *from2++;
    *to = '\0';
}
/*
 * get letter
 *   li      -> letter index
 *   f       -> copy to file
 */
getlet(li, f)
register FILE *f;
{
    register int i;
    register char *s;
    char buf[BUFSIZE], lastc, ch;
    int n, j;
    long k;
    int num;

    fseek(tmpf, let[li].adr, 0);
    k = let[li+1].adr - let[li].adr;
    if(k)
        k--;
    for(i=0; i<k; i++) {
        s = buf;
        num = ((k-i) > sizeof(buf)) ? sizeof(buf) : (k-i);
        if((n = read(tmpf->_file, buf, num)) <= 0)
            return(FALSE);
        lastc = buf[n-1];
        if(write(f->_file, buf, n) == ERROR)
            return(FALSE);
        i += n;
        fflush(f);
    }
}
```

```
    }
    if( lastc != EOF) {
        read(tmpf->_file, buf, 1);
        write(f->_file, buf, 1);
    }
    return(TRUE);
}

/*
 * create smail lock file
 */
lock(file)
char *file;
{
    register int f, i;

    if (locked == LOCKON)
        return;
    scat(curlock, file, smaillock);
    for (i=0; i<10; i++) {
        if((f = creat(curlock, MODE)) != ERROR){
            close(f);
            locked = LOCKON;
            return;
        }
        sleep(2);
    }
    unlock();
    unlink(smailtmp);
    exit(2);
}

unlock()
{
    unlink(curlock);
    locked = LOCKOFF;
}

setsig(i, f)
int i;
int (*f)();
{
    signal(i, f);
}

done()
{
    unlock();
    unlink(smailtmp);
    exit(1);
}
```

```
/*
 *   des.c   6/16/86
 *
 *   This function is a variation of the
 *   system password function which implements
 *   the Federal DES standard. It takes as
 *   input a key, text and a crypt or decrypt
 *   flag. To encrypt, it receives 8 byte multiples
 *   of text and it sends back 11 bytes of
 *   ascii characters in the set [a-zA-Z0-9./].
 *   To decrypt, it receives multiples of 11 bytes
 *   to decrypt and it returns the clear text.
 */

#include "tables" /* System DES permutation Tables */
void setkeyx(); /* System DES subroutines */
void encryptx(); /* setkey and encrypt */

char *des(key, text, decrypt)
char *key;
char *text;
int decrypt;
{
    char static block[66];
    static char keyblock[66];
    static char out[512];
    char iobuf[16];
    register int i, j, c;

    for(i=0; i < 66; i++)
        keyblock[i] = 0;
    /*
     *   initialize output buffer
     */
    for(i=0; i < 512; i++)
        out[i] = 0;
    /*
     *   break 8 bytes of the key into
     *   a 64 bit block, ignore the
     *   most significant bit
     */
    for(i=0; (c= *key) && i < 64; key++)
    {
        for(j=0; j < 7; j++, i++)
            keyblock[i] = (c >> (6-j)) & 01;
        i++;
    }
    /*
     * System DES implementation
     */
}
```

```
setkeyx(keyblock);

while(*text)
{
    /*
    /*    initialize buffers
    */

    for(i=0; i<16; i++)
        iobuf[i] = 0;
    for(i=0; i < 66; i++)
        block[i] = 0;
    if (decrypt)
    {
        /*
        /*    convert back to original character set
        /*    and decode the 11 bytes back to 8 bytes.
        */
        for(i=0; (c= *text) && i < 64; text++)
        {
            if ( c > 'Z')
                c -= 6;
            if ( c > '9')
                c -= 7;
            c = '.';
            for(j=0; j < 6; j++, i++)
                block[i] = (c >> (5-j)) & 01;
        }
        for(i=0; i < 48; i++)
            E[i] = e2[i];

        /*
        /*    call the actual DES algorithm
        */
        encryptx(block, decrypt);

        /*    convert the 64 bit block
        /*    back to 8 bytes of clear
        /*    text
        */
        for(i=0; i < 8; i++)
        {
            c = 0;
            for(j=0; j < 8; j++)
            {
                c <<= 1;
                c |= block[8*i+j];
            }
            iobuf[i] = c;
        }
        strncat(out, iobuf, sizeof(iobuf));
    }
}
```

```
    }
    else
    {
        /*
        /*      create a 64 bit block
        */
        for(i=0; (c= *text) && i < 64; text++)
        {
            for(j=0; j < 8; j++, i++)
                block[i] = (c >> (7-j)) & 01;
        }

        for(i=0; i < 48; i++)
            E[i] = e2[i];

        encryptx(block, decrypt);
        /*
        /*      create 11 6 bit words from
        /*      a 66 bit block
        */
        for(i=0; i < 11; i++)
        {
            c = 0;
            for(j=0; j < 6; j++)
            {
                c <<= 1;
                c |= block[6*i+j];
            }

            /*
            /*      limit the character set
            */

            c += '.';
            if (c > '9')
                c += 7;
            if (c > 'Z')
                c += 6;
            iobuf[i] = c;
        }
        strncat(out, iobuf, sizeof(iobuf));
    }
}

return(out);
}

void
setkeyx(key)
char *key;
{
    register int i, j, k;
    int t;
```

```
/*
 * First, generate C and D by permuting
 * the key. The low order bit of each
 * 8-bit char is not used, so C and D are only 28
 * bits apiece.
 */
for(i=0; i < 28; i++) {
    C[i] = key[PC1_C[i]-1];
    D[i] = key[PC1_D[i]-1];
}
/*
 * To generate Ki, rotate C and D according
 * to schedule and pick up a permutation
 * using PC2.
 */
for(i=0; i < 16; i++) {
    /*
     * rotate.
     */
    for(k=0; k < shifts[i]; k++) {
        t = C[0];
        for(j=0; j < 28-1; j++)
            C[j] = C[j+1];
        C[27] = t;
        t = D[0];
        for(j=0; j < 28-1; j++)
            D[j] = D[j+1];
        D[27] = t;
    }
    /*
     * get Ki. Note C and D are concatenated.
     */
    for(j=0; j < 24; j++) {
        KS[i][j] = C[PC2_C[j]-1];
        KS[i][j+24] = D[PC2_D[j]-28-1];
    }
}

}

/*
 * System encrypt subroutine
 */

void
encryptx(block, edflag)
char    *block;
int     edflag;
{
    int    i, ii;
    register int t, j, k;
```

```
/*
 * First, permute the bits in the input
 */
for(j=0; j < 64; j++)
    L[j] = block[IP[j]-1];
/*
 * Perform an encryption operation 16 times.
 */
for(ii=0; ii < 16; ii++) {
    /*
     * Set direction
     */
    if(edflag)
        i = 15-ii;
    else
        i = ii;
    /*
     * Save the R array,
     * which will be the new L.
     */
    for(j=0; j < 32; j++)
        tempL[j] = R[j];
    /*
     * Expand R to 48 bits using the E selector;
     * exclusive-or with the current key bits.
     */
    for(j=0; j < 48; j++)
        preS[j] = R[E[j]-1] ^ KS[i][j];
    /*
     * The pre-select bits are now considered
     * in 8 groups of 6 bits each.
     * The 8 selection functions map these
     * 6-bit quantities into 4-bit quantities
     * and the results permuted
     * to make an f(R, K).
     * The indexing into the selection functions
     * is peculiar; it could be simplified by
     * rewriting the tables.
     */
    for(j=0; j < 8; j++) {
        t = 6*j;
        k = S[j][(preS[t+0] < 5)+
                (preS[t+1] < 3)+
                (preS[t+2] < 2)+
                (preS[t+3] < 1)+
                (preS[t+4] < 0)+
                (preS[t+5] < 4)];
        t = 4*j;
        f[t+0] = (k > 3) & 01;
        f[t+1] = (k > 2) & 01;
        f[t+2] = (k > 1) & 01;
```

```
        f[t+3] = (k>>0)&01;
    }
    /*
    * The new R is  $L \wedge f(R, K)$ .
    * The f here has to be permuted first, though.
    */
    for(j=0; j < 32; j++)
        R[j] = L[j] ^ f[P[j]-1];
    /*
    * Finally, the new L (the original R)
    * is copied back.
    */
    for(j=0; j < 32; j++)
        L[j] = tempL[j];
}
/*
* The output L and R are reversed.
*/
for(j=0; j < 32; j++) {
    t = L[j];
    L[j] = R[j];
    R[j] = t;
}
/*
* The final output
* gets the inverse permutation of the very original.
*/
for(j=0; j < 64; j++)
    block[j] = L[FP[j]-1];
}
```

```
/* System Permutation Tables used by DES algorithm */

/*
 * Initial permutation
 */
static char IP[] = {
    58,50,42,34,26,18,10, 2,
    60,52,44,36,28,20,12, 4,
    62,54,46,38,30,22,14, 6,
    64,56,48,40,32,24,16, 8,
    57,49,41,33,25,17, 9, 1,
    59,51,43,35,27,19,11, 3,
    61,53,45,37,29,21,13, 5,
    63,55,47,39,31,23,15, 7,
};

/*
 * Final permutation
 */
static char FP[] = {
    40, 8,48,16,56,24,64,32,
    39, 7,47,15,55,23,63,31,
    38, 6,46,14,54,22,62,30,
    37, 5,45,13,53,21,61,29,
    36, 4,44,12,52,20,60,28,
    35, 3,43,11,51,19,59,27,
    34, 2,42,10,50,18,58,26,
    33, 1,41, 9,49,17,57,25,
};

/*
 * Permuted-choice 1 from the key bits
 */
static char PC1_C[] = {
    57,49,41,33,25,17, 9,
    1,58,50,42,34,26,18,
    10, 2,59,51,43,35,27,
    19,11, 3,60,52,44,36,
};

static char PC1_D[] = {
    63,55,47,39,31,23,15,
    7,62,54,46,38,30,22,
    14, 6,61,53,45,37,29,
    21,13, 5,28,20,12, 4,
};

/*
 * Sequence of shifts used for the key schedule.
 */
static char shifts[] = { 1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1, };
```

```
/*
 * Permuted-choice 2
 */
static char PC2_C[] = {
    14,17,11,24, 1, 5,
    3,28,15, 6,21,10,
    23,19,12, 4,26, 8,
    16, 7,27,20,13, 2,
};

static char PC2_D[] = {
    41,52,31,37,47,55,
    30,40,51,45,33,48,
    44,49,39,56,34,53,
    46,42,50,36,29,32,
};

/*
 * The C and D arrays used to calculate the key schedule.
 */

static char C[28];
static char D[28];
/*
 * The key schedule.
 * Generated from the key.
 */
static char KS[16][48];

/*
 * Set up the key schedule from the key.
 */
/*
 * The E bit-selection table.
 */
static char E[48];
static char e2[] = {
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9,10,11,12,13,
    12,13,14,15,16,17,
    16,17,18,19,20,21,
    20,21,22,23,24,25,
    24,25,26,27,28,29,
    28,29,30,31,32, 1,
};

/*
 * The 8 selection functions.
 */
static char S[8][64] = {
```

14, 4,13, 1, 2,15,11, 8, 3,10, 6,12, 5, 9, 0, 7,
0,15, 7, 4,14, 2,13, 1,10, 6,12,11, 9, 5, 3, 8,
4, 1,14, 8,13, 6, 2,11,15,12, 9, 7, 3,10, 5, 0,
15,12, 8, 2, 4, 9, 1, 7, 5,11, 3,14,10, 0, 6,13,

15, 1, 8,14, 6,11, 3, 4, 9, 7, 2,13,12, 0, 5,10,
3,13, 4, 7,15, 2, 8,14,12, 0, 1,10, 6, 9,11, 5,
0,14, 7,11,10, 4,13, 1, 5, 8,12, 6, 9, 3, 2,15,
13, 8,10, 1, 3,15, 4, 2,11, 6, 7,12, 0, 5,14, 9,

10, 0, 9,14, 6, 3,15, 5, 1,13,12, 7,11, 4, 2, 8,
13, 7, 0, 9, 3, 4, 6,10, 2, 8, 5,14,12,11,15, 1,
13, 6, 4, 9, 8,15, 3, 0,11, 1, 2,12, 5,10,14, 7,
1,10,13, 0, 6, 9, 8, 7, 4,15,14, 3,11, 5, 2,12,

7,13,14, 3, 0, 6, 9,10, 1, 2, 8, 5,11,12, 4,15,
13, 8,11, 5, 6,15, 0, 3, 4, 7, 2,12, 1,10,14, 9,
10, 6, 9, 0,12,11, 7,13,15, 1, 3,14, 5, 2, 8, 4,
3,15, 0, 6,10, 1,13, 8, 9, 4, 5,11,12, 7, 2,14,

2,12, 4, 1, 7,10,11, 6, 8, 5, 3,15,13, 0,14, 9,
14,11, 2,12, 4, 7,13, 1, 5, 0,15,10, 3, 9, 8, 6,
4, 2, 1,11,10,13, 7, 8,15, 9,12, 5, 6, 3, 0,14,
11, 8,12, 7, 1,14, 2,13, 6,15, 0, 9,10, 4, 5, 3,

12, 1,10,15, 9, 2, 6, 8, 0,13, 3, 4,14, 7, 5,11,
10,15, 4, 2, 7,12, 9, 5, 6, 1,13,14, 0,11, 3, 8,
9,14,15, 5, 2, 8,12, 3, 7, 0, 4,10, 1,13,11, 6,
4, 3, 2,12, 9, 5,15,10,11,14, 1, 7, 6, 0, 8,13,

4,11, 2,14,15, 0, 8,13, 3,12, 9, 7, 5,10, 6, 1,
13, 0,11, 7, 4, 9, 1,10,14, 3, 5,12, 2,15, 8, 6,
1, 4,11,13,12, 3, 7,14,10,15, 6, 8, 0, 5, 9, 2,
6,11,13, 8, 1, 4,10, 7, 9, 5, 0,15,14, 2, 3,12,

13, 2, 8, 4, 6,15,11, 1,10, 9, 3,14, 5, 0,12, 7,
1,15,13, 8,10, 3, 7, 4,12, 5, 6,11, 0,14, 9, 2,
7,11, 4, 1, 9,12,14, 2, 0, 6,10,13,15, 3, 5, 8,
2, 1,14, 7, 4,10, 8,13,15,12, 9, 0, 3, 5, 6,11,

};

/*

* P is a permutation on the selected combination
* of the current L and key.

*/

static char P[] = {
16, 7,20,21,
29,12,28,17,
1,15,23,26,
5,18,31,10,
2, 8,24,14,

```
        32,27, 3, 9,  
        19,13,30, 6,  
        22,11, 4,25,  
};  
  
/*  
 * The current block, divided into 2 halves.  
 */  
static char L[32], R[32];  
static char tempL[32];  
static char f[32];  
  
/*  
 * The combination of the key and the input, before selection.  
 */  
static char preS[48];
```

DESIGNING AND IMPLEMENTING A NETWORK AUTHENTICATION SERVICE
FOR PROVIDING A SECURE COMMUNICATION CHANNEL

by

CHRISTOPHER P. CHANCE

B. S. E. E., University of South Carolina, 1979

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

Designing and Implementing a Network Authentication Service
for Providing a Secure Communication Channel

by

Christopher P. Chance

An Abstract of a Master's Report

The purpose of this paper is to provide an overview of an authentication service that was developed to implement a secure communication channel within a computer network. This service provides a "secret" electronic mail facility that verifies the identities of the communicating parties and preserves the integrity of the message contents. The design, implementation, and possible enhancements are discussed. A review of existing authentication services is also given. Appendices contain user manual pages and a copy of the implementation code.

The system utilizes conventional encryption methods for providing authenticity of messages sent either locally or remotely. The service was designed using the concept of servers. Three servers encompass the system. The Authorization server provides the user interface to the system. It enables a user to enroll in the service and establishes the protocol between host machines. The Name Server is the interface between the Authorization Server and the registration database. The Message Server provides the user interface to the "secret" mailboxes.