CONFLICTS IN PROVIDING SECURITY IN
COMPUTER-BASED MESSAGE SYSTEMS

by

ANTHONY JOSEPH KUSALIK
B.Sc., The University of Lethbridge, 1978

THIS THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in
THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
August 1982

© Anthony Joseph Kusalik, 1982
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date Aug 25, 1982
Abstract

This thesis identifies and discusses problems which develop when guaranteed message security is to be incorporated into the design of a computer-based message system. The conflicts are the result of following existing security and message system design models, the basic one-way nature of the communications provided, and the desire to provide customarily expected functionality. Solutions to some of these problems are presented.

A message system model is given, and an implementation following the model is described and evaluated.

The ideas of researchers primarily concerned with computer system or network security differ from those in the area of computer-based message system design. Views vary on the correct location within a message system of facilities for data encryption, delivery confirmation, and duplicate message suppression. These differences of approach are discussed.
# Table of Contents

Abstract

Table of Contents

List of Figures

Acknowledgements

1. Introduction
   1.1 Overview
   1.2 Objectives
   1.3 Outline
   1.4 Computing Environment
   1.5 Terminology

2. Research Background
   2.1 Threats to Security
   2.2 Overview of Cryptosystems
   2.3 Key Distribution and Management
   2.4 Secure Computing Environment
   2.5 CBMS Model
   2.6 Placement of Encryption Protocol
   2.7 Summary

3. Conflicts and Incompatibilities
   3.1 Message Time Integrity
   3.2 Signatures Despite Key Compromise
   3.3 Retraction of Posted Messages
   3.4 Protection with Multiple-Destination Messages
3.5 Proof of Delivery ........................................ 43
3.6 Summary .................................................. 45

4. Towards Solutions .......................................... 46
   4.1 Message Time Integrity ................................. 46
   4.2 Sealing for Multiple-Destination Messages .......... 48
   4.3 Remaining Conflicts ................................. 50

5. Implementation ............................................ 52
   5.1 CBMS .................................................. 52
   5.2 Cryptographic Facility ................................ 54

6. Results and Evaluation ................................... 56

7. Summary and Conclusions ................................ 60

8. List of References ........................................ 62

Appendix A: DES Implementation ........................... 67
List of Figures

Figure 1  Cryptographic Information Flow . . . . . . 9
Figure 2  CBMS Model . . . . . . . . . . . . . . 22
Figure 3  Sealed Object . . . . . . . . . . . . . . 49
Acknowledgements

This work would not have been possible without the moral support, patience, and love of my wife Dolores, the encouragement of David Etherington and Gerald Neufeld, and the monetary support of the Natural Sciences and Engineering Research Council. I would like to thank David Cheriton for originally suggesting the ideas which motivated my interest in this area. Credit is also due to those faculty members and fellow students who contributed their ideas and criticism during discussions, especially Gerald Neufeld, John Demco, and Steve Deering.
1. Introduction

This thesis identifies and discusses some problems and conflicts which arise in the design of a computer-based message system. These conflicts are the result of attempting to guarantee the security of transferred messages, providing customarily expected functionality, and following software models which advocate the separation of the message transfer service from security mechanisms and message preparation / management facilities.

1.1 Overview

Since their inception computer-based message systems have become a popular feature on computer networks and distributed timesharing systems due to their convenience and usefulness. A computer-based message system (CBMS) is any combination of hardware and software which provides distribution, storage, and interactive manipulation of one-way communications (messages) between applications or users on a computer network. The "electronic mail" or "computer mail" facilities commonly present on computer systems are examples of CBMS's.

Clients of these systems are often concerned with secrecy, integrity, and authenticity of messages. Typically, the sender wishes assurances that a message cannot be disclosed or disrupted en route, while the recipient is primarily concerned that the message is not fraudulent [Simmons 79]. Except for certain sensitive environments, such as the military, banking, and government, most CBMS designs address this issue in cursory terms.
Techniques utilizing data encryption do exist for providing various types of protection for communications between two principles on the same or different machines [Diffie & Hellman 76, Needham & Schroeder 78, Popek & Kline 79, Kent 81]. State-of-the-art CBMS designs and specifications [Birrell et al. 81, Deutsch 81] often incorporate provisions for use of these techniques. However, actual incorporation of these techniques into the design of a CBMS can entail problems.

With the design and implementation of any significant system of software, there exist a set of goals. Sample goals may be following a certain programming philosophy, specific functionality of the end-product, style of the man-machine interface (its "user-friendliness"), security and protection of data, maintainability, and efficiency. Often completely satisfying the entire set of goals is impossible; conflicting goals are discovered.

1.2 Objectives

The main intent of this thesis is to describe problems encountered when incorporating guarantees of message transport security into the design of a CBMS. It is also the aim of this work to provide and evaluate solutions to some of these problems. An amount of background information is necessary for the discussion, namely descriptions of the characteristics of the computing environment being assumed, threats to network security, cryptosystems, provision of security by use of cryptosystems, and models for user security and CBMS design.
Further, the placement of security mechanisms in the CBMS model requires description and justification. Another objective is to test certain models and ideas presented by actual implementations, and to evaluate the results. The final objective is to formulate and present ideas for future work.

1.3 Outline

The remainder of this section outlines the computing environment which serves as the basis for discussion. Section 2 describes pertinent work in the fields of cryptography and network data security. Security and CBMS design models are also described. A discussion of the problems which arise comprises section 3, solutions to some of which are given in section 4. Section 5 contains a description of software implemented to test certain ideas presented in section 2. Section 6 evaluates this software, as well as the models and assumptions of section 2. Finally, section 7 offers a summary of the work and indicates further areas of investigation.

1.4 Computing Environment

The emphasis in computer architectures is toward computer networks, collections of computers linked by a transmission medium and software which allows communications between peer entities on different hosts. Individual networks may be connected by gateways to form larger networks or internetworks. A distributed computer system is a heterogeneous or homogeneous computer network which supports distributed applications and communications between (user)
processes on the same or separate machines. Many familiar long-haul and local-area networks such as ARPANET and Ethernet. This thesis assumes an underlying computer system of such a distributed nature.

The network provides the only available means of low-delay communication. However, the transmissions are insecure and susceptible to surreptitious activity. High-delay but secure channels of varying bandwidth, such as personal contact, trusted courier, or armoured truck do exist.

The computer system serves an academic or closed-shop community. Users are interested in reliable communication software, such as a CBMS. However, only a small percentage of traffic requires security guarantees. Further, the number of perpetrators of malicious activity is small, and they are interested in a small subset of total message traffic. (The greater the activity of an intruder, the greater the probability of his discovery and apprehension. If the amount of pernicious activity makes reliable service too costly or uncertain, users will simply employ other means to communicate.)

Each user has access to a secure and inexpensive encryption / decryption facility. It is implemented as some combination of hardware and software. The bandwidth of the facility need not approach that of the communication medium, but should not overly impede the user (several thousand bits per second is not unreasonable [Denning 79, Michelman 79]).
1.5 Terminology

In the research literature, it is common to have the term "message" refer to an arbitrary unit of communication exchanged between any two network peer entities, and also to the unit of communication transferred from sender to receiver by a CBMS. Many lower-level "messages" of the former type may be necessary to transfer the latter type of "message"; the latter type is a special case of the former. The intended meaning of the word, when used, is usually clear from context. In the case of ambiguity, "message" refers to the more general case, while the phrase "mail message" or "CBMS message" indicates the more restrictive meaning.
2. Research Background

This section is a survey of work in pertinent areas of network security and cryptosystems. A CBMS design model and a model for a user's secure computing environment are given. A discussion on best placement of encryption protocols within communication software levels is presented. The section concludes with a summary of the environment and the models assumed.

2.1 Threats to Security

An enemy or intruder can pose a threat to a user of a computer network in the following ways:
(a) determination of message content,
(b) alteration of some portion of a message,
(c) impersonation of the user responsible for a message, and
(d) disruption of service.
Circumvention of these threats can be considered to consist of six distinct components:
(a) secrecy (ensuring that information is disclosed only to specific users),
(b) privacy (preventing an intruder from determining the source, destination or length of a message, or transmission frequency, i.e. "traffic analysis"),
(c) integrity (ensuring that destruction or modification of data is detected),
(d) authentication (preventing the forgery of information),
(e) availability (ensuring that access to information cannot be maliciously interrupted), and
(f) confinement or containment (preventing leakage of information by a service program by subtle or covert means [Lampson 73]).

For the purposes of this paper, the terms "security" and "protection" include the ideas of secrecy, integrity, authentication, and determination of disruption of service.

Providing assurances of privacy, availability, and confinement are more difficult [Lampson 73, Michelman 79, Padlipsky et al. 79, Popek & Kline 79, Kent 81]. For example, there does not appear to be any practical means of preventing an enemy severing a communication link or injecting "noise" sufficient to disrupt transmissions. An example of the confinement problem is two processes communicating by affecting each other's data throughput. Controlling such covert channels appears to require software verification [Padlipsky et al. 79, Popek & Kline 79].

Thwarting traffic analysis is of secondary importance in many applications [Kent 81]. As such, it is not included in the idea of security, but is addressed in its own right in subsequent sections.

The current sophistication of the microelectronics industry provides the intruder with the tools to accomplish the following clandestine activities [Popek & Kline 79]:

(a) Tapping of lines. It is regarded as a simple matter to record information passing through a given network medium without detection.

(b) Introduction of spurious messages. It is often possible to introduce messages into an operating network in such a
manner that they pass all relevant consistency checks as genuine.
(c) Retransmission of previously transmitted legitimate messages (given the previous two abilities).
(d) Selective alteration of messages or prevention of message delivery.

Security mechanisms attempt to circumvent these types of activities.

Threats to secure communication are not restricted to an active, external intruder. Network software could deliberately or erroneously send a message to an incorrect destination, modify portions of messages, deliver duplicate messages, or lose messages ("trojan-horse" software).

To provide protection, there must exist some trusted group of entities, operations, or objects. Reducing the number of these reduces the size of the problem of determining, proving, and maintaining the security of the system.

2.2 Overview of Cryptosystems

Encryption is a method of manipulating information into a form which is unintelligible without a specific, often secret, item of information, the key. Encryption can be thought of as a function which transforms plaintext or cleartext (the original information) into ciphertext or a cryptogram (incomprehensible information) under the control of a key. Decryption is the inverse operation, mapping ciphertext into plaintext, again under the control of a key. Encryption is the act of encrypting or enciphering; decryption is the act of
decrypting or deciphering. A cipher is a family of encryption transformations. A cryptosystem or cryptographic system is the union of a cipher and the family of its inverse decryption transformations. The key is the parameter which specifies an individual transformation. A cryptographic facility is an implementation of a cryptosystem. Figure 1 depicts information flow in a computer system with such a facility.

For a cryptosystem to be useful, the mappings must not be trivial or easily deduced; the degree to which the key is necessary in determining the mappings between cleartext and cryptograms indicates the "strength" of the cryptosystem.
The objective of a cryptoanalyst is to analyze and break the cryptosystem, i.e. to produce the best estimate of the original cleartext given full knowledge of the encryption and decryption schemes, access to ciphertext, and a variety of side information (language statistics, general context of the on-going conversation, probable words, etc.). It may be possible for the cryptoanalyst to produce meaningful plaintext without having the decryption key in certain cases. However, breaking a cryptosystem usually means finding a scheme that is capable of determining the decryption key, or equivalently the cleartext, for any ciphertext given enough available information [Lempel 79].

A desireable property of cryptosystems is that with the change of only one bit of cleartext or ciphertext, the probability of any bit of the resulting cryptogram or plaintext changing is one-half [Popek & Kline 79]. This property is implicitly the cryptosystem's ability to mask statistical properties such as character groupings of the cleartext. As a result, "strong" cipher techniques are excellent mechanisms for error detection and, when used in conjunction with redundant information such as checksums, for ensuring data integrity.

A cipher can be thought of as a finite automaton. A stream cipher encrypts information bit-by-bit: the transformation of a specific bit is dependent on preceding information and hence the state of the automaton. With a block cipher, plaintext is broken into blocks of a specific length, usually the length of the key, and each block is enciphered independently, irrespective of the information in preceding
blocks and hence the state of the automaton. Certain block cipher techniques can be used to provide a stream cipher [NBS 75].

Block ciphers are preferred in computer applications due to synchronization problems with stream ciphers [Popek & Kline 79]. For example, an error at a certain point in a stream cipher is propagated through subsequent operations (unless the correct state for a given input is reestablished [Diffie & Hellman 79]). Also, the update of a selected portion of (encrypted) data requires the reencryption of all subsequent data.

There are two major classes of cryptosystems. In single-key cryptosystems, also called symmetric, conventional, or classical cryptosystems, the encryption and decryption functions use the same key. The NBS Data Encryption Standard [NBS 77] is a classical cryptosystem. With asymmetric cryptosystems the two keys are not identical.

A public-key cryptosystem, first described by Diffie and Hellman [1976], is a special type of asymmetric cryptosystem. Such a system must have three properties:

(a) Given the encryption key, $K$, it is very difficult to determine the decryption key, $K'$.
(b) The encryption and decryption algorithms must be easy to compute.
(c) Let $E$ denote the encryption function, $D$ the decryption function, and $P$ a plaintext message. Then

$$P = D(E(P,K),K').$$
With this cipher technique the encryption key can be publicly disclosed without risking a security breach. A user has a pair of corresponding keys, one secret and one public. A further property is usually stipulated, namely that

\[ E(D(P, K'), K) = P = D(E(P, K), K'). \]

This refinement enables the use of public and secret keys interchangeably for encryption (and decryption). This property is useful in both key distribution and message authentication (as is shown in following sections). Proposed public-key cryptosystems are based on known NP-complete problems such as the knapsack problem [Merkle & Hellman 78], factoring of very large numbers into the product of primes (the RSA algorithm) [Rivest et al. 78], and computing logarithms modulo a prime number [Diffie & Hellman 76]. Implementations of the RSA technique exist [Michelman 79].

The distribution of keys with the two types of cryptosystems is discussed in the next section.

The security of an encryption algorithm is measured as the difficulty or amount of "work" necessary in deducing the decryption key given specific information. This forms the basis for the following categorization of cryptoanalytic attack:

(a) Ciphertext-only. The cryptoanalyst has detailed knowledge of the encryption and decryption algorithms, samples of ciphertext, and in the case of public-key encryption, knows the encryption key.

(b) Known-plaintext. In addition to the information for ciphertext-only attack, the cryptoanalyst has matching
plaintext / ciphertext pairs.
(c) Chosen-plaintext. In addition to the information for known-plaintext attack, the cryptoanalyst can obtain ciphertext corresponding to plaintext of his choice. Cryptosystems must be resistant to known-plaintext attack to be considered useful [Diffie & Hellman 79]. (A cryptoanalyst can guess probable plaintext portions of a message, such as system greetings or probable words. This allows use of known-plaintext techniques to break a system safe only from ciphertext-only attack.)

The strength of a cryptosystem is related to the ratio of the length of the key to the length of the data. A cryptographic system is "perfectly secure" if all that can be discerned from a cryptogram is that it exists and the length of the corresponding cleartext [Gifford 82]. Such ciphers require keys that are as long as the data they encode. Often, breaking a more practical cryptosystem is a matter of economics [Diffie & Hellman 76], a trade-off between memory and computation time [Hellman 80]. A cryptographic system is called computationally or practically secure if, even when enough information is theoretically available to break the system, the amount of computation to do so is realistically unattainable. Since the theories of computational complexity cannot presently prove the difficulty of such a problem (is P=NP?) [Lempel 79], cryptography is forced to rely on a "certification" process. A cryptosystem is certified secure if it withstands a concerted cryptoanalytic assault under circumstances favorable to the cryptoanalyst. It is a misconception that public-key cryptosystems are more
secure than symmetric systems; the security of both is based on computational work factors [Simmons 79].

2.3 Key Distribution and Management

The basis of security for two communicating parties is the secrecy of a private key, for

"There is little security in a system in which people happily broadcast their decryption keys for other users as often and whenever they like" [Michelman 79].

To have a secure network conversation, the participants must obtain matching keys to encrypt and decrypt transmitted information. The security of the keys is essential during distribution and the lifetime of their use. A matched pair of keys forms a logical channel on the network [Popek & Kline 79]. Possession of an appropriate key admits one to the channel; without it the channel is unavailable.

The consideration of key distribution has led to the identification of the key management problem:

"Keys must be produced and distributed not once, but constantly. In some systems they must be changed with the passage of time, or with the amount of traffic, and in all systems they must be changed when they are feared compromised ... Keys must be provided to new users of the system and old keys must be retired as users withdraw" [Diffie & Hellman 79].

Key management becomes more difficult with increases in the number of potential communicants.

The secure transmittal of keys over a secure channel such as a trusted courier is relatively expensive and time-consuming. What is required is a method for interchanging keys securely over the network. Needham and Schroeder [1978] outline a set of protocols for establishing secure
communication between network entities using conventional or public-key cryptosystems. Various extensions to enhance the degree of protection have followed [Popek & Kline 79, Denning & Sacco 81, Booth 81].

Needham & Schroeder's protocols utilize a trusted entity: a centralized key distribution facility called an authentication server (AS). Each user registers a key with the AS by some secure means (trusted courier, for example). This key becomes the basis for establishing communications with this user. To initiate network communication, users must acquire a shared conversation key in a single-key system, or each other's public keys in a public-key system.

The essential step in setting up a secure communication channel is for the initiator to generate a message with two properties:

(a) it must be comprehensible only to the intended recipient, i.e. allow only the recipient to use its contents to identify himself to the initiator;
(b) it must be evident to the recipient that the message originated with the initiator.

Some of the information passed during this initiation step can then be used to protect the remainder of the conversation.

For the purposes of discussion, assume an entity $S$ wishes to converse with an entity $R$ over the network. Let $I_x$ denote a nonce identifier attributed to entity $x$. ("Nonce" means "used only once".) Let $\{ d \}^K$ denote datum $d$ encrypted using key $K$ and let

$$X \rightarrow Y : m$$
signify entity X sending message m to Y.

The objective of the protocol in a single-key cryptosystem is to have the two parties safely obtain a new, unique, truly random key CK to protect the subsequent conversation. Let KS and KR represent S's and R's secret keys, respectively, registered with the authentication server. The steps of the protocol are:

(S1) S -> AS : S, R, IS

Upon receipt, the AS looks up KS and KR and generates CK for the ensuing conversation. This message need not be enciphered since S can confirm that the AS received the correct information in reply (S2).

(S2) AS -> S : { IS, R, CK, [ CK, S ]KR }KS

Because this message is encrypted with KS, only S can discover CK. S checks for the presence of R and IS in the reply to ensure that an intruder did not modify (S1) and to guarantee its time integrity, i.e. that it is not a reply of a previous response. S retains CK.

(S3) S -> R : { CK, S }KR

Only R can decrypt this message and obtain CK. R knows the identity of the intended correspondent as authenticated by the AS. At this point, S knows that any communication it receives encrypted with CK must originate with R and any communication encrypted with CK it emits can only be understood by R. R would be in a similar state if (S3) could be proven not to be a replay of a previous message.

(S4) R -> S : { IR }CK
(S5) S -> R : { f(IR) }CK
"f" is some agreed upon function such as subtraction by one. After the exchange of these two messages, R can be sure all messages it receives encrypted with CK are not replays.

The use of encryption in messages (S1) through (S5) ensures their integrity and secrecy. Subsequent messages can be protected by enciphering with CK and, if necessary, seriating using $I_R$ as a seed.

The number of steps in the protocol can be reduced to three by maintenance of a secure cache of items of the form $R: CK, \{ CK, S \}^{KR}$ for common destinations. The exchanges of the protocol would then be:

(S3') $S \rightarrow R : \{ CK, S \}^{KR}, \{ I_S \}^{CK}$
(S4') $R \rightarrow S : \{ f(I_S), I_R \}^{CK}$
(S5) $S \rightarrow R : \{ f(I_R) \}^{CK}$.

The added information is required by S to be sure of the time integrity of replies from R.

In a public-key cryptosystem, the object of the protocol is to have each participant obtain the other's public key in a secure manner to initiate a protected conversation. Let PKx and SKx represent the public and secret keys, respectively, of entity x. PKS and PKR are registered with the AS. Each user knows PKAS. The steps of the protocol in this case are:

(P1) $S \rightarrow AS : S, R$

This message need not be enciphered since S can confirm that the AS received the correct information in reply

(P2).
(P2) AS -> S : [ PKR, R ]SKAS

By this exchange S determines R's public key. The integrity of PKAS (as stored by S) is critical to prevent impersonation of the AS by an intruder. Message (P2) is enciphered to guarantee its authenticity and integrity, not its secrecy: to prevent the substitution of the public key of some miscreant for PKR.

(P3) S -> R : [ IS, S ]PKR

This can only be understood by R, so S can be sure of R's identity.

(P4) R -> AS : R, S

(P5) AS -> R : [ PKS, S ]SKAS

This exchange is analogous to (P1), (P2).

(P6) R -> S : [ IS, IR ]PKS

This message can only be understood by S. The inclusion of IS assures S that the message is not a reply.

(P7) S -> R : [ IR ]PKR.

This proves the time integrity of the messages to R.

The secrecy of (P2) and (P5) is not important since any user can request public keys. Integrity and secrecy of messages (P3), (P6), and (P7) is guaranteed by use of encryption. After this key exchange, then, each user has the other's public key. Four of the steps, (P1), (PK), (P4), and (P5) can be eliminated by each user having a local cache of public keys.

Subsequent messages in the dialogue cannot simply be enciphered by PKS or PKR since any user can obtain these keys and so inject messages into the stream. Two possible solutions are to doubly encrypt each message, i.e.
or to use $I_S$ or $I_R$ as the basis for the seriation of encrypted message blocks.

The message sequence $(S4), (S5)$ is referred to as a "handshake". Message sequence $(S3'), (S4'), (S5)$ and sequence $(P3), (P6), (P7)$ are referred to as "two-way" or "double" handshakes. The primary purpose of a handshake is to authenticate one principle to the other - to ensure that messages being received are not replays from a previous conversation.

The protocols do not prevent traffic analysis; the intended correspondents are obvious in messages $(S1), (P1)$, and $(P4)$.

The two protocols are very similar and require nearly the same number of message exchanges. They both establish a secure communications channel. Therefore, the choice of a cryptosystem is better based on the economy and strength of the encryption algorithm, rather than on protocol complexity [Needham & Schroeder 78, Popek & Kline 79].

Should a key become compromised (modified or determined by unauthorized individuals) without its user's knowledge, no protocol using this key can initialize a secure communication channel. It is assumed, therefore, that users fostering these protocols store all keys securely and are vigilant for evidence of key compromise. Immediately upon detected compromise, it is the user's responsibility to take corrective action such as reporting the fact to the AS. (The threat posed by a compromise is restricted once corrective action is begun.) The
user is held responsible for any communications authenticated by the key in question up to the time of such a report. Periodic key changes irrespective of known compromise are well-advised as they decrease the potential harm of unperceived or future key compromise and limit the amount of time and ciphertext an enemy has available to break the system. Procedures to facilitate changing a user's registered key or the key(s) of the authentication server exist [Michelman 79].

2.4 Secure Computing Environment

User isolation, each user on his own private and physically isolated machine, is a sound basis for security [Denning 79, Rushby 81, Gifford 82]. Each user is considered to have a secure computing environment if a computer model is adopted where all users and individual network subsystems (file server, for example) are physically isolated from one another. If more than one user entity exists on a specific host, each will be assumed to have a secure computing environment if the operating system maintains this model. That is, the kernel of a shared host must provide environments and communication channels between them such that individual components cannot distinguish such a shared situation from a physically distributed one.

The cipher facility is contained in or part of the user's secure environment, since the communication medium between users is not trusted. The facility may be single-key or public-key, and be provided by hardware, software, or some combination of these two.
In this model, threats to security occur only with communications outside each isolated environment. Since all access to a user's environment is through an easily defined and small number of points (the user's terminal and the network interface, for example), it is much easier to provide security by implementing safeguards "at" the access points. To safeguard the use of the network for mail message transfer, file storage, and database management, for example, all susceptible information is encrypted before being placed on the communications medium. (There will of course be instances when encryption is not necessary, such as protocol step (P1)).

2.5 CBMS Model

The model for the computer-based message systems to be used in this study is based on, but not identical to, that devised by the North American Systems Environment Subgroup of IFIP WG 6.5 [Schicker 79, Deutsch 81, IFIP 81a] and Study Group VII of the CCITT [CCITT 80].

The clients of a CBMS are called users. Users engage in one-way communications, where the source of the communication is the originator, or sender, and the destination is the recipient, or receiver.

A message is the basic unit of information and is composed of an envelope and its contents. The message contents may be data of any form, for example, text, facsimile data, or executable programs, and is the information that an originator wants to communicate to a recipient. An envelope has a specific, predefined representation and format.
A user agent (UA) is a functional entity which assists a user in preparation, inspection, or management of messages, or acts on the user's behalf to receive and send messages. A user
agent is usually a multi-functional CBMS-specific program such as MSG [Vittal 81a], though a more general purpose program such as an editor or data management program may under some implementations be used as a UA. Received messages are stored in a file or file structure belonging to the user and dedicated to this purpose.

The message transfer system (MTS) accepts a message from an originator's UA and the responsibility to deliver it to the intended recipient. The MTS may perform routing, storage, and encryption functions to accomplish this task. The message transport system can be subdivided into a number of interconnected entities called message transfer agents, or MTA's. The MTA's are distributed among the hosts on the network.

To send a message, an originator, via a UA, engages in a posting protocol with the MTS, or more specifically, with the posting message transfer agent (for this transmission). Transferring message contents from an originator's user agent to the MTS is called posting. The point at which responsibility for the message passes from the originator's UA to the MTS is called the posting slot. The posting MTA constructs an envelope for the message, and forwards the message towards its destination within the MTS. The message is routed or relayed toward its destination by other MTA's until received by the MTA "closest" to the recipient's host. This MTA, the delivery message transfer agent, will engage in a delivery protocol with a UA for the recipient. Delivery is complete when the message contents have been transferred from
the message transfer system to the recipient's user agent. The delivery slot is the point at which the recipient's UA accepts responsibility for the message from the MTS.

Each message is identified by a (globally) unique identifier, the message identifier, which is stored in the envelope and generated at the time of posting by the posting MTA.

The envelope of a message consists of information for the use of the MTS, such as the message's final destination, instructions to invoke special MTS services, its route through the message transport system, or billing information. Some of the information may be redundant with information in the message contents. A recipient's UA may obtain the envelope for a message from the delivery MTA before the message has passed through the delivery slot.

The MTS handles message contents as bit-stream data. No attempt is made to interpret or modify any portion. Agreement on format and interpretation of message contents is the responsibility of the users involved in the communication.

Encryption and decryption of messages is provided by a trusted software component of the user's (secure) environment, such as a subroutine or program, accessible to the UA. The actual encryption or decryption operation may be performed by a combination of software, hardware, or firmware.

A service provider (SP) is an entity which performs operations other than message transfer which are employed by UA's and MTA's, such as message format translation and permanent storage. Service providers are not necessarily
internal components of the CBMS. They are more often free-standing system components which offer their services to other system components. The authentication server described earlier is a service provider.

One SP, the directory server (DS), provides directory information and name validation. Each user registers with the DS a unique name by which to be referred, his network mail name. Part of this name is an indication of the host computer to which his mail is to be sent: the host computer supporting the delivery MTA for messages destined to this user. The DS supports the definition of a group name as a set of mail names. Expansion and verification of group name is also facilitated by the DS.

As part of the posting protocol, the sender of a message must indicate the identity of the intended recipient(s) by some combination of group and network mail names. The posting MTA validates and expands these names to determine the network addresses to which messages should be routed. If multiple recipients are specified, the MTA creates copies of the message contents, and each is enclosed in an envelope and treated as a separate message. The sender may obtain the message identifier of the message or each message copy as part of the protocol.

When a (new) message is ready to be delivered, the delivery MTA may notify the user asynchronously, if the user's host operating system provides such a facility. A UA can always poll the delivery MTA about the existence of new messages. After successful delivery, a record of the message may be retained by the MTS, but no copies of the contents are
kept.

The protocols outlined in section 2.3 may be used to authenticate a UA and MTA to each other, and initiate a "connection" for posting or delivery. If security requirements are not as stringent or fulfilled at other levels, conventional password authentication of the UA or no measures at all may be considered. Security is discussed further in the next section.

The message transfer service is built upon the services of lower-level network protocols, following a layered architecture [Zimmerman 80]. The MTS is built upon the network session level [IFIP 81b, NBS 81]; an MTA is addressable as a process on a particular network host. The UA must incorporate support of this level of protocol to accomplish delivery and posting. Above this level, the UA must be able to interpret the form and representation of the message contents.

2.6 Placement of Encryption Protocol

The services of the CBMS are built upon a hierarchy of network protocol layers. It is necessary to establish at what level encryption protocols should be applied to provide for the security of messages. The debate is motivated by varying views of the trustworthiness that can be attributed to the software of a network communications subsystem.

The simplest point at which encryption can be applied in a network situation is at the link level. All information received and transmitted by a node is encrypted. Key management is straightforward: keys are maintained in firmware
only for adjacent nodes and installed by low-bandwidth means. Such placement alone does not provide confinement [Padlipsky et al. 79, Kent 81] or protect against accidental or deliberate compromise of data at higher levels. Information cannot be encrypted selectively, so that the cryptographic facility must be very efficient so as not to impede network throughput.

Messages can be encrypted at the highest level possible, in this case above the level provided by the MTS. This placement is supported by an "end-to-end" argument which contends that even if encryption is provided at a lower level, the sender and receiver are still required to utilize their own safeguards at the highest level to guarantee security:

"The function in question [encryption or delivery confirmation] can completely and correctly be implemented only with the knowledge and help of an application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement)" [Saltzer et al. 81].

For succeedingly lower network levels, the number of identifiable entities tends to decrease. For example, the link level between two hosts may be supporting a number of higher-level connections. Key distribution and the establishment of secure channels is therefore easier with low-level encryption. The degree to which cryptographic facilities are shared is increased also.

Providing communication safeguards at higher network levels reduces the amount of software whose correct functioning must be assured [Popek & Kline 79]. A distributed application requires the correct and secure functioning of all intermediate
levels before it can rely on lower-level facilities. Two communicating high-level peer entities do not require guarantees on the correctness of the link level, or any lower level, if they implement their own set of safeguards.

Price [1981] presents an argument for situating encryption at the transport level, but does concede that greatest security is achieved by end-to-end placement.

The advantages of locating data encryption at the endpoints of the communication facility include:

(a) no user is forced to be satisfied with an algorithm (implemented at a lower level) which he feels is too weak,
(b) the user can change keys or encryption methods whenever he feels the existing one has been compromised,
(c) varying degrees of protection are permissible,
(d) the responsibility for security is relieved from the communication service and placed with the user.

End-to-end encryption may be more efficient since only sensitive data need be enciphered. Also, a significant amount of network line traffic is protocol overhead. End-to-end placement minimizes the amount of such overhead that must be enciphered.

Saltzer et al. [1981] contend that

"... if data transmission subsystems perform encryption and decryption, they must be trusted to manage securely the required encryption keys, the data will be in the clear and thus vulnerable as it passes into the target node and is fanned out to the target application, and the authenticity of the message must still be checked by the application. If the application performs end-to-end encryption, it obtains its required authentication check, it can handle key management to its satisfaction, and data is never exposed outside the application."
This argument would suggest that to guarantee security, encryption and decryption of messages and key management be done at a level above that provided by the MTS: cleartext being available only within the user's secure environment. The implications for a CBMS of such positioning are that the contents of a message are encrypted prior to posting and that a recipient can receive a message without necessarily knowing (during the delivery protocol) the appropriate decryption key [Needham & Schroeder 78]. Situating encryption protocols at this level is allowed by the NBS message format standard [Deutsch 81].

Designers of computer-based message systems often assume that the message transfer facility can be "trusted" [Schicker 81]. Guaranteeing the operation of a MTS requires software verification, a difficult task [McCcauley & Drongowski 79]. As greater functionality is provided by an MTS [NBS 81], it becomes no easier. The literature does not contain any descriptions of verified message transfer software. The MTS is therefore not presumed secure or correct, though to attract clients and be considered useful, its services are reliable.

If message security was to be provided by the MTS, the key management problem would have to be addressed. To minimize the threat of key compromise, each MTA would have a distinct key, rather than one for the entire MTS. This complicates the coordination of MTA activity to allow key changes. Corrective action in the case of compromise would require exceptional measures and human intervention since automated procedures are dependent on the security of the key in question and an
intruder with the compromised key may impersonate the legitimate MTA and initiate a key change. Further, all messages handled by the MTS would be vulnerable in such circumstances, whereas if an end-to-end scheme were used, compromise of a user's key would only jeopardize communications involving that user.

Security guarantees provided by the MTS would also require additional encryption and decryption operations, incurring greater cost. Enciphering and deciphering the message would be required at posting, between each "step" in the message's path through the MTS, and at delivery.

The other advantages to the end-to-end scheme include simplifying the design of the transfer service and freeing those accountable for the MTS from major legal responsibility should a security breach occur. Requiring all messages to be transferred securely by the MTS would be needless and inefficient since only a minority are assumed to require protection. Treating this minority as a "special case" would only complicate MTA software.

End-to-end facilities may be costly if, for example, a large number of delivered messages must be discarded because of disruption by an intruder. Lower-level mechanisms may have diagnosed the activity earlier and saved the burden of complete transport of the messages. It is assumed, however, that the probability of any particular message being modified or fraudulent is small, that such disruption is not the norm.
End-to-end arguments can also be applied to other services often associated with the MTS such as delivery verification. Whether or not the MTS delivers a message is as important as presentation of that message to the user. Any manner of disaster may have befallen the UA after delivery (disk crash, power failure, etc.). It is reasonable, then, to locate such a service within the user agent. For example, some field in the message contents can be used to communicate the desire for confirmation from sender to receiver. The recipient's UA composes and sends a deliver confirmation message automatically and immediately upon the recipient's "reading" of the message. Such confirmation not only indicates that the message was delivered (as in MTS-level placement), but additionally that the message has been "given" to the user.

2.7 Summary

The following is a summary of the assumptions being made for the discussion to follow.

The environment for computer users is a distributed computer system composed of hosts on a single network or an internetwork incorporating a number of networks connected by gateways. The network is vulnerable to malicious assault. Each user is provided with a secure environment by having his own physically isolated personal computer, or being supported by a shared host which emulates such a situation. A classic or public-key cryptographic facility which is certified against known-plaintext attack is provided with the user's secure environment. A trusted authentication server which supports
the protocols outlined in section 2.3 is present.

The CBMS present is composed of a transfer facility and message preparation / management programs. The MTS is not guaranteed secure or correct, but does provide highly reliable services. The message transfer mechanism is separate from the security system. Guarantees of message security and delivery confirmation are provided by end-to-end methods. The MTS may still duplicate the services for efficiency or convenience.

Secure use of the CBMS is threatened by an enemy having sophisticated tools but only concerned with a small fraction of the total traffic. Further, users deem it necessary to encrypt a minority of all the messages handled by the CBMS.
3. Conflicts and Incompatibilities

With most computer software systems there exist conflicts and incompatibilities when attempting to satisfy a set of design or implementational goals. In the case of computer-based message systems, there exist conflicts between providing message protection, following a specific design model, and attempting to provide a functionally "acceptable" product. Five such conflicts are developed in this section: ensuring the time integrity of mail messages; providing digital signatures despite key compromise; protecting a message, copies of which are to be delivered to multiple disparate destinations; providing proof of message delivery; and retraction of posted messages given a digital signature facility.

Necessarily, the logical units of communication in a CBMS are uni-directional, or one-way, as opposed to two-way communications as may exist over a network "connection" or occur in a "conversation". A communication is one-way if its originator is only concerned that it reaches the prescribed destination; the sender does not wait for an acknowledgement. Additionally in the case of a CBMS, there may be a significantly long delay between the origination and receipt of a communication. This characteristic of messages causes problems when security is to be ensured.

Needham and Schroeder [1978] outline a set of end-to-end protocols that utilize cryptosystems to provide protection of two-way communications in a network by securely establishing a dialogue between the two principles. Enhancements to these
protocols have been suggested by others [Denning 79, Popek & Kline 79, Booth 81, Denning & Sacco 81]. Analogous guarantees are desirable with one-way communications of a CBMS.

3.1 Message Time Integrity

In a secure network transportation subsystem, it is desirable to prevent an intruder from recording a transmission, and later repeating or reintroducing it. That is, the time integrity of each communication should be guaranteed.

The protocols of Needham and Schroeder assure time integrity in two-way communications by relying on the ability of each of the two participants to take part in a "handshake" (section 2.3). This handshake is not possible for one-way communications.

It may be proposed that the CBMS be designed so that as part of the posting and delivery protocols, the sender and receiver, respectively, take part in such an encryption handshake. The operation of sending or receiving a message would not be logically complete until the handshake was completed. Maintenance of the required "connection" could be handled by a level within the MTS, with the user agent taking part in the protocol asynchronously. This approach is not attractive because:

(a) The mail system may take an exceptionally long time to deliver the message, possibly due to network link or host failures, or slow or congested communication lines. In particular, the delay between posting and delivery need
only be greater than the amount of time that the two parties are willing to wait for the handshakes to be completed. This may also impact performance, since the messages could not be considered through the delivery slot until the handshake is complete, necessitating the possible maintenance of many connections.

(b) The intended recipient may not be active when the message system is ready to deliver the message; in fact the recipient may not be active for an indeterminate amount of time. (It is usually infeasible to have all users active on an interactive timesharing system at once). Creation of a user process on attempted delivery of a message is a possibility, but allows a security breach, since the entity which creates the user process must be able to pass the test for the authenticity of the user. Therefore, the process and the program it initiates would have to be verified correct. Such verification is not easy to do, unfortunately [McCauley & Drongowski 79].

Needham and Schroeder recognize the problems inherent with one-way communication and propose the use of "timestamps" as one way of eliminating the need for a handshake. They propose that for either single-key or public-key cryptosystems a timestamp be associated with (the encrypted contents of) a message indicating the time of sending. The resolution of the time standard used must be fine enough to allow differentiation of any two messages from the same source. All recipients maintain a database with entries of the form {source,timestamp}. 
Also associated with each recipient are values $T$, $dt_1$, and $dt_2$. The value $dt_1$ is an upper bound on asynchrony between the recipient's local clock and all other clocks on the network. Value $dt_2$ is an upper bound on the delay between a source sending a message and its arrival within the recipient's secure environment. The value $T$ is the sum of $dt_1$ and $dt_2$. When a message arrives, it is rejected if a duplicate \{source,timestamp\} entry exists in the database, or if its timestamp predates the local time by more than $T$. The size of the database is minimized by keeping entries no older than amount of time $T$.

As Needham and Schroeder point out, the value of $T$ may need to vary if a message may only arrive in a recipient's security environment when he is present, since $dt_2$ may increase in such a case. In addition $T$ may vary due to variations in clock synchronization and in network transport delays.

Several problems exist with this scheme, including the difficulties of maintaining a time standard [Dickson 80] and determining a discretionary value in a memory / error trade-off. A local clock, and hence a timestamp from that clock, may be erroneous due to human error (for example an operator setting an incorrect time), a hardware malfunction, or the efforts of an intruder. The smaller the value of $T$, the smaller the size of register, but the greater the probability of rejecting a legitimate message with an "old" timestamp.

The values $dt_1$ and $dt_2$ can be based on maximal, average, or "desirable" values. Decreasing the magnitude of $T$ correspondingly decreases the amount of memory and work to
maintain the register. No matter what value is chosen for $T$, it is always conceivable that a message will arrive and be rejected even though it is legitimate. The degree to which this would occur corresponds inversely to the magnitude of $T$. A legitimate message with an exceptionally old timestamp may have been delayed in delivery by severe congestion, network routing errors, or network link failure.

The intention of an intruder may be to disrupt a message by preventing its delivery. The intruder may be able to accomplish this by taking advantage of the protocol in a subtle way and ensuring that the message is sufficiently delayed or compromising the accuracy of the timestamp. He may be able to do this innocuously by being the source of a congesting amount of network traffic. Under such circumstances it would be harder to prove malicious intent than if the intruder was caught directly severing or impeding communications.

3.2 Signatures Despite Key Compromise

Needham and Schroeder outline methods for providing evidence sufficient to prove to a third party that a particular message is exactly as received from a particular sender. This is commonly known as providing a "digital signature". All discussion concerning digital signatures will presume the use of a public-key cryptosystem. (Use of single-key schemes for this facility is much less popular in the literature).

One outlined scheme for providing such evidence, first described by Diffie and Hellman [1976] is to doubly encrypt the message to be sent from the sender, $S$, to the recipient, $R$,
The credibility of the signature is based on PKS as registered with the authentication server. If knowledge of PKS allows decipherment (after decryption with SKR), then SKS must have been used to encipher the message. Only S knows SKS; the message must have originated with S.

It is pointed out that the value of the protocol is dependent on S not changing his key pair. Otherwise it is necessary that the authentication server maintain a record of old public keys and the time of each change, and for signed messages to contain the time they were signed. However, suspect key compromise normally requires a key change. Also, periodic key replacements are expected as they limit the degree of potential loss due to future key compromise. Maintenance of such a key database is an undesirable requirement to make of the AS.

Popek and Kline [1979] suggest the following scheme for providing digital signatures. The originator of a message, S, "signs" the message (by encrypting with his secret key) and sending it to an individual serving as a "notary public" (NP). The NP appends a timestamp, signs the entire message with its secret key, and sends the result back to S. The originator can add appropriate cleartext information and send the message to the intended recipient, R:

\[ S \rightarrow R : \{ S, PKS, \{ \text{message } \}^{SKS}^{SKNP} \}^{PKR}. \]

The receiver tests the validity of the signature and (securely) stores
S, PKS, \{ \{ \text{message} \}^{SKS}, \text{timestamp} \}^{SKNP}.

Implicit with this approach is that the recipient trusts the contents of an important message only if the sender is willing to include the signature. The trustworthiness of the signature is independent of S later changing his key pair, but is dependent on the credibility of the notary public. If necessary, several separately notarized copies of the message can be sent. A variation is to have the signature of the NP be of the form:

\{ \{ \text{message} \}^{SKS}, \text{timestamp}, PKS \}^{SKNP}.

Another solution requires the receiver, upon receipt of a doubly encrypted message,

\{ \{ \text{message} \}^{SKS} \}^{PKR}

to decipher it with his secret key and send

\{ \text{message} \}^{SKS}

to the authentication server. The AS replies with:

AS -> R : \{ \{ \text{message} \}^{SKS}, \text{timestamp}, PKS \}^{SKAS}.

R can confirm this "notarized" signature if necessary, then store it. R can never forge the signature. The AS need only store current public keys and a record of its key pairs and time of changes. If S should later change his key pair, R need not be concerned, or even know. The signature could also be supplied by an NP. This solution was proposed by Booth [1981], though without the inclusion of the timestamp in the signature.

The later two techniques are very similar, and differ mainly in the time at which a signature is obtained. Both approaches could be used with two-way communications. In a
CBMS, use of Booth's technique is unwise as R may not receive the message for a significant amount of time, long enough for S to have changed his key pair for legitimate reasons. In such a case, the reply from the AS would not provide a valid signature, being of the form:

\[
\{ \{ \text{message} \}^{SKS}, \text{timestamp}, PKS' \}^{SKAS}
\]

where PKS' is S's new public key. In fact, R would be unable to decipher the original message, since

\[
\{ \{ \text{message} \}^{SKS} \}^{PKS'}
\]

will almost certainly not yield the original cleartext. This results from the fact that R is required to (be able to) act within a short time of S sending the message.

3.3 Retraction of Posted Messages

It is a contentious point whether an originator can exercise any control on a message after it is posted but before it is delivered, that is, when it is the responsibility of the MTS. In particular, there does not exist a definitive body of opinion on allowing or providing retraction of posted messages (by the sender) prior to delivery. Some MTS designs include the ability to selectively retract messages [NBS 81]. However, since the MTS is not guaranteed secure, retraction by an end-to-end mechanism is considered. Typically, such retraction would be desirable in the case of key compromise and would not be selective.

Suppose message retraction is not allowed. The first digital signature described in section 3.2, and consequently Booth's technique, exhibits the following characteristic: after
a user has signed a message by doubly encrypting each block and successfully posting it, he can still prevent delivery in an end-to-end fashion by changing his key pair prior to the delivery of the message to the recipient. In particular, the recipient would receive a message of the form

\[ \{ \{ \text{message} \}^{SKS} \}^{PKR} \]

which could not be deciphered with S's new public key, say PKS'.

If R has S's (old) public key in his cache, and does not update the entry before attempting decryption, the cleartext can be obtained. Therefore, the algorithm for updating a cache entry could attempt to use an existing entry, ignorant of any changes made known to the AS, and only update the entry if decryption with the old key resulted in nonsensical plaintext.

Consider the result of this algorithm in the following scenario: A user suspects, and rightly so, that an enemy has determined his secret key and so changes his key pair. The enemy forges a (signed) message using the old secret key. The unsuspecting recipient trusts the authenticity of the message since it can be successfully decrypted using the old public key. The impersonated user is not responsible for any ramifications of the counterfeit message, since he has changed his key pair (registered with the AS) prior to the fraudulent message being sent.

Such an algorithm should therefore not be used to prevent message retraction. In fact, there appears to be no way to prevent retraction if authenticity of messages is to be maintained using the first (or last) protocol outlined in
section 3.2.

If the protocol outlined by Popek and Kline is used to provide digital signatures, there is no means by which end-to-end message retraction can be provided. Otherwise it would be necessary for the NP to change his key pair, and perjure himself as to his old key pair and the time of change. Such action would quickly lead to legal action, or at least CBMS users questioning the trustworthiness of the NP.

3.4 Protection with Multiple-Destination Messages

The encryption protocols outlined in section 2.3 all involve communication between two parties. However, with a CBMS it is often the case that a single source wishes to send duplicate copies of a message to several different destinations. The design of many mail systems recognizes this characteristic of use [Birrell et al. 82, Landweber & Solomon 82, Neufeld 82] and allow a sender to specify a list of intended recipients for a message. After completing the posting protocol, the MTS attempts to send a copy of the message to each receiver given. The sender is involved in a single posting protocol for multiple recipients: a great convenience. Suppose that the message contents are encrypted in end-to-end fashion. It is not clear what key is to be used. Having all the intended recipients use the same secret key is undesirable. The only obvious solution is to allow users to be in predefined groups where each member has knowledge of the separate group key (with a symmetric cryptosystem) or key pair (with a public-key system). The
message can then be encrypted using the recipients' group key or public key. Disadvantages to this approach are the logistics of distributing the group key(s) securely to all members, the restriction that sender can only send to predefined groups, and the principle that the more people that know a key, the greater the security risk.

Without this feature, the sender is forced to post the messages individually, each encrypted with the appropriate (dissimilar) key. The number of messages routed and delivered by the MTS would be the same. The sending operation would be less efficient for the originator and posting MTA, as there would be less opportunity for caching techniques to be applied.

3.5 Proof of Delivery

The idea of confirmation of delivery is a common one in the design of mail systems. Typically, a CBMS can indicate whether a particular message has been successfully delivered to the intended recipient, that is, provide the originator with information about the outcome of the transmission of a message through the transfer system. Delivery confirmation is the converse of the idea of digital signatures; instead of providing the recipient with proof of what was received, the sender is given proof that the receiver obtained what was sent. The article of proof cannot be disavowed by the sender or the receiver, respectively, and should be sufficient to satisfy an impartial third-party.
With digital signatures, a trusted entity is used, either the AS or a NP, and the credibility of a signature is dependent on its trustworthiness. For delivery confirmation, such a trusted entity is again desired and the MTS is an obvious candidate. Unfortunately, the transfer facility is not assumed protected or correct, so that any confirmations it supplies may be suspect.

End-to-end measures may be utilized. The problem with an end-to-end protocol is that the recipient may actually have received a message but refuses to acknowledge it.

Suppose that delivery confirmation is provided by both the MTS and end-to-end methods. A user may receive an affirmative confirmation from the transfer facility, but no end-to-end acknowledgement. In this case, the user may question the reliability of the MTS or the intended recipient, but cannot assume the message delivered.

Schicker [1981] outlines twelve levels of service which could be provided by the MTS and still be termed delivery confirmation, and sketches algorithms to implement some of them. He further outlines methods for providing third-party proof of delivery confirmation by the MTS. However, the credibility of all these services is dependent upon the trustworthiness of the message transfer facility. The best that such a "feature" may provide is a notification that is almost certain but not guaranteed, or coupled with end-to-end measures, an indication of suspect activity.
3.6 Summary

The one-way nature of mail messages (and the potential delay between posting and delivery), the requirement of end-to-end encryption and security guarantees, and the desire to provide certain user-friendly features are all at least partly responsible for the five problems. Network software systems must always address the problem of delays, whatever the cause, but typically the entities at the endpoints of the communication exist simultaneously. With computer-based message systems this is not the case and the magnitude of delay to be tolerated is much greater.

Solutions to these problems are addressed in the next section.
4. Towards Solutions

In the preceding section five areas of contention between various CBMS design and implementation goals were discussed. Each conflict can be overcome to some degree simply by removing one of the requirements responsible, or by accepting a certain degree of error as inevitable but tolerable. In one case, a resolution is possible that fulfills all the requirements of the CBMS design.

4.1 Message Time Integrity

Section 3.1 outlined the message time integrity problem and a proposed solution. However, the solution does not insure that messages which have been delayed in delivery for legitimate reasons are accepted, while always rejecting replayed messages. The solution involves a trade-off between the absurdness of the decision to accept or reject, and the size of the database holding comparison \{source,timestamp\} entries. No suggestion for an appropriate value of T is given in the literature.

It seems reasonable to assume that $dt_1$, the bound on clock asynchrony, can be set to several minutes [Dickson 80]. The value of $dt_2$ is required to be much larger. It is conceivable that message delays of up to 12, 24, or more hours are commonplace in the MTS. For example, with CSNET [Landweber & Solomon 82], hosts which must "connect" to the transport mechanism by telephone may be the source of such delay. An MTA host may be down for several days, delaying any messages which are "trapped" within the node or messages whose only possible
communication paths require that node.

A value of one week for T seems appropriate, with the option that a user can temporarily increase it due to an absence of greater duration (vacation, for example). Several hundred encrypted messages over a one week period is a reasonable expected maximum. (Twenty to fifty messages in total per day is common on ARPANET [Crocker et al. 79].) A database with this number of entries does not require an inordinate amount of effort or resources to maintain. An increase in database size due to a user's absence is considered an exception, a temporary phenomenon. Also, it is expected that if a MTA is inoperative for over one week, special measures would be taken to recover or re-route any "trapped" messages. It is unlikely that messages would be delayed for more than this amount of time for any other reason. Therefore, the number of legitimate messages rejected by this scheme should be "acceptably" small.

Note that this value for T necessitates that users are frequently present on the system: once or twice per week is insufficient. Otherwise the following situation may occur: A message is sent on a Monday to user R. The message is delayed and is not obtained by the delivery MTA until Friday. However, user R only "logs on" to the system once a week, say every Thursday. In this situation, the message would be discarded, though it was legitimate. For such users, the value of T must increase as the average frequency of use decreases. The expected size of the database must increase accordingly.
4.2 Sealing for Multiple-Destination Messages

Section 3.3 identifies a problem with providing end-to-end encryption for a message with multiple, distinct recipients. Gifford [1982] proposes the idea of using cryptosystems to "seal" data objects with a key. As objects are arbitrary sequences of bytes in the scheme, it can be applied to CBMS messages. A brief description of cryptographic sealing is provided, followed by a discussion of how it can be applied to the problem at hand.

A "sealed object" is composed of encyphered data and information for verification and "unsealing". Sealed objects are self-authenticating, and in the absence of an appropriate set of keys, only provide information about the size of their contents. Keys are the basic unit of security and authentication in the mechanism. New keys can be freely created at any time or "derived" from existing keys using "key operators". Associated with each key is a unique identifier. When data is encrypted with a key, the identifier of the key that will decrypt it is maintained with the data.

Consider some cleartext CT and some collection of keys Kl ... Kn. This technique allows the construction of a sealed object of the form illustrated by figure 3. The object is encyphered, as suggested by an enclosing box, and marked with the key that can be used to decrypt it. IKEY is a newly created, truly random key of a classical cryptosystem and Checksum(CT) is the result of a checksum operation on CT. The checksum later provides verification of successful decryption. The cleartext can be a message, or a signed message as
described in 3.2 with the exception that IKEY is used in place of PKB. To "unseal" the object, a recipient must possess a key which will allow one of the items of the opener to be unsealed, i.e. one of the keys Kl ... Kn or a key which will unseal an object containing one of Kl ... Kn. IKEY is not retained by the sender after use.

![Diagram of Sealed Object](image)

Figure 3
Sealed Object

Cryptographic sealing can be used for a message destined for multiple receivers under both single or public-key cryptosystems. Minor enhancements to the AS may be necessary. With a single-key system, the originator of a message supplies to the authentication server a list of intended recipients for a message. The AS returns an opener such as illustrated in figure 3, as well as IKEY, where Kl ... Kn are the secret keys of the recipients. In the case of a public-key cryptosystem, an identical operation can be followed, except that Kl ... Kn are the public keys of the intended recipients, or the sender can construct the opener, querying the AS for any keys which are not in his cache of public keys. IKEY is used to seal the
message. The opener and enciphered message can now be sent as a sealed message.

Upon receipt of a sealed message by any of the intended recipients, each can attempt, using his secret key(s), to unseal each item of the opener. The mechanism can quickly determine the legitimacy of each key because of the use of key identifiers. Only the intended recipients can unseal an item of the opener and hence unseal the message.

Cryptographic sealing, then, provides a mechanism to post a message to be delivered to multiple recipients and still provide end-to-end protection. The group of recipients need not be predefined. Further, the mechanism is efficient for both the sender, recipients, and MTS.

4.3 Remaining Conflicts

There are no apparent resolutions to the other three conflicts discussed in section 3.

The unacceptability of Booth's signature technique in section 3.2 results from the potential time delay inherent with mail messages. The use of the protocol would require that the AS record for some period the old public keys and the time of each key change, as well as the current public key, for each user. Further, an encrypted message would require a timestamp, and be of the form

\[ \{ \{ \text{timestamp} \}^{SKAS}, \{ \text{message} \}^{SKS} \}^{PKR}. \]

This would allow the recipient to decipher the message as well as obtain a signature, providing that the message was sent within the time period for which keys are retained. This is
clearly a memory / error trade-off. As in section 4.1, suppose that messages can be delayed up to one week in the MTS. If it is assumed that a user changes his key pair once per week on average, this at least doubles the size of the database supported by the AS. In the case of message time integrity, the costs of expanding the database to accommodate temporary absence or infrequent use of the system are born by the user in question. In this case, such cost would be an additional burden on a shared system resource, the AS. As a result, such an approach to allow use of Booth's technique is deemed not acceptable. No other modifications to allow its use are obvious, save those which alter the basic character of the protocol.

Section 3.3 discusses problems with message retraction: neither signature technique given in section 3.2 both allows and disallows end-to-end retraction. The choice of signature technique dictates whether retraction is possible, and vice versa. A solution to this problem, a scheme for providing digital signatures irrespective of the ability to retract posted messages, is not obvious and may not exist.

In the case of delivery confirmation, there is little that can be done about a recipient not participating in an end-to-end delivery confirmation protocol. A trusted independent entity can be utilized, though the nature and location of such an entity is not clear. (The MTS cannot be used since it is assumed not verified or secure.)
5. Implementation

To test the soundness of the computer-based message system model presented (section 2.5), an implementation was performed on the Verex Operating System [Lockhart 79]. This operating system, developed from the Thoth Operating System [Cheriton 79], is part of the research facilities at the University of British Columbia.

Unfortunately, Verex does not provide the model of a secure computing environment as described in section 2.4. The operating system consists of a kernel and privileged processes. The kernel is minimal, supporting creation, scheduling and destruction of processes, and synchronous communication by messages among processes. The operating system is not verified secure and there exist privileged, trusted processes to accomplish "system tasks".

5.1 CBMS

The message system implemented closely follows the model described in section 2.4 with the following exceptions:

(a) The system only incorporates a single host. This restriction is largely due to the inavailability of a network link to hosts which provide adequate accessibility and resources. Therefore the MTS is composed of a single (local) MTA. The Verex host machine includes local-area network hardware and such a network is under development.

(b) No directory server exists on the system. As a result, mail names for users are the user names supported by the operating system and the ability to use group names is not
provided to users, though the supporting software exists within the MTA. (Specification of multiple recipients is supported by the MTS.)

(c) No authentication server exists on the system. The MTA relies on the "login" procedure of the operating system to ensure the authenticity of users. When a process communicates with the MTA, the identity of the corresponding user as recorded by the operating system is obtained and assumed correct. Validity of mail names is provided by the user name validation mechanism of Verex.

The system is noteworthy in at least one respect. The MTA is implemented as a Verex file server and supports the Verex I/O Protocol [Cheriton 80a]. This protocol is standard among all data input / output facilities (e.g. disk file server, printer server, terminal server). Therefore, nearly any program which does file input / output (including system programs) can be used (to some degree) as a UA. A user can, for example, compose a message with an editor and post the message by simply specifying an output file name of the string "$mail/" concatenated with the recipient's user name.

The CBMS is independent of the fact that each user is not provided a secure computing environment; if this was the case, few, if any, modifications to the CBMS would be necessary.

The MTS provides posting and delivery services. In the posting protocol the sender provides the names of one of more intended recipients and the message contents. The MTA verifies the given recipient names. When a new message "arrives" and a process of the receiver exists on the system, a notification of
new mail is sent to the user's terminal via a privileged process. A recipient can obtain new messages immediately after their arrival. A UA can always query the MTS about the existence of new mail messages. The recipient obtains the message contents, and optionally the envelope, during the posting protocol. No delivery indication (success or failure) is returned to the sender; "best effort" delivery is provided by the MTS. No explicit duplicate suppression exists.

A user agent program is written which exploits all the features of the CBMS. The UA provides a user interface to maintain a collection of messages, send messages to one or more recipients, retrieve new messages, and output messages. The UA does not attempt to interpret the message contents and treats it as byte-stream data. Format interpretation is at the discretion of the user.

5.2 Cryptographic Facility

The DES algorithm [NBS 77] was implemented on the Verex Operating System as a set of relocatable functions which can be included in application programs. The software provides encryption and decryption of blocks of 64 bits of information. The facility has a bandwidth of about 175 bits per second. The correct operation of the software was verified using published tests and results [Gait 77]. Appendix A contains a source listing of the functions.

To allow Verex users easy use of the DES algorithm, a set of interface functions and encryption / decryption application programs were also written. The encryption program allows
users to encrypt byte or computer-word input using an eight-character key. The output can be either as computer words (in groups constituting 64 bits) or as character representations of the hexadecimal values of these words. The decryption program accepts either type of encryption output and provides the original cleartext, bytes or computer words. Since no authentication server exists, key distribution is accomplished by direct exchange of keys between intended correspondents.

The implementation of the cryptographic facility allows users to encrypt and post a message in one operation by use of "pipes" (a familiar construct of UNIX). Similarly, a recipient can decrypt a message by "piping" the output from the UA to the decryption program. The motivation for this approach is that if a decryption / encryption facility is provided as part of the UA, the generalized encryption and decryption program would still be required for other applications, such as encrypting data to be stored by the file system. The use of "pipes" allows encryption and decryption of messages via the more general mechanism.
6. Results and Evaluation

This section evaluates the preceding CBMS implementation and considers some enhancements. Alternatives to several ideas and assumptions already presented are given and discussed.

The CBMS implementation described has been used as the exclusive message facility available to Verex users for over one year. During that period many modifications have been made to the user interface of the UA. Several changes have been made to the MTS and UA to improve their performance and utilization of system resources. However, the basic design of the CBMS has remained the same with little user criticism of it.

The acceptance of the CBMS design (implicated by lack of criticism) may not be as indicative of its soundness as first appears. The host system is a small research facility for a small group of friendly users, mainly students, with few, if any, security concerns. The dissimilarity with more real-life situations may obscure potential shortcomings or errors in the ideas presented.

The treatment of message contents as bit-stream data by the UA allows the CBMS to be treated as a communication sublayer. For example, an application program may create data in a specialized format and "pipe" it to the UA for posting. Upon receipt, a user can "pipe" the message from the receiving UA to the corresponding application. Also, such a philosophy allows "active" messages [Vittal 81b] to be sent: the messages transferred by the CBMS may be executable programs. Upon receipt, they could be output to a file and executed.
The principle that the MTS provide reliable "best efforts" rather than guarantees to deliver messages simplifies the software implementation. In general, allowing for all possible circumstances (rather than ignoring or handling at a different level situations and events that are possible but highly unlikely) greatly complicates software. The end-to-end philosophy for message transfer guarantees does simplify the MTS implementation as claimed in section 2.7.

Given the existence of the DES cryptographic facility, and assuming that the privileged processes of the Verex Operating System are secure and correct, an authentication server could be installed, and the protocols outlined in section 2.3 utilized to provide secure key management and distribution. Support of the protocols could be added to the general-purpose encryption / decryption programs already present or message protection could be incorporated into the UA. The MTS could be augmented to use the AS to authenticate potential clients.

The MTS (one local MTA in this case) is designed to use a directory server for mail name expansion and validation. Since no such facility exists, the MTA relies on the user name validation service of the operating system and cannot support group names as recipient names. However, should a DS become available, the only required modification to the transfer facility would be redirection of the requests for mail name expansion and validation.

The implemented UA could be modified to support end-to-end delivery confirmation as discussed in section 2.6. Such an addition would not require any design changes to the CBMS.
The requirement of end-to-end guarantees motivates several of the problems outlined in section 3. For example, since MTA's exist as "sessions" on their respective hosts, they can participate in two-way communications to bring about message transfer. Hence problems with message time integrity would be (potentially) solved. A UA would authenticate himself to a posting MTA and post a message by two-way communication. The message would be routed securely by MTA's using secure conversations at each step. Finally, the message could be delivered using a protected two-way connection. Guaranteeing message security would still require verification of the correctness of the MTA, though the user community may be willing to trust unverified software and accept a low level of threat rather than facing the message time integrity problem.

Mail messages are one-way only due to the timeframe being considered. They can be thought of as datagrams supporting a yet higher level (than the OSI application layer) n-way communication between two or more users where, for example, the "conversation identifier" is a certain subject.

The end-to-end placement of security measures allows prevention of "traffic analysis". For example, "nop" or "dummy" messages could be sent to an arbitrary set of recipients to mask communications with a certain user. A "nop" message would contain arbitrary data and be ignored by its recipient. Determination of receipt of such a message would require decryption; it must be impossible for an enemy to differentiate between "nop" and legitimate messages. The sender would be billed for the transfer of all such messages.
However, if "nop" messages became commonplace, the costs for a recipient to receive, decypher, and discard each one may become significant and require administrative action.

With the techniques outlined in section 3.2 for signing messages, the amount of information that must be signed can be reduced by determining a characteristic value, CS, of a message and having the CS replace the entire message in the signature. (A characteristic function maps from the domain of all possible messages to a much smaller range with the property that given a message and its image under the function, it is computationally difficult to find another message with the same image) [Needham & Schroeder 78].

A closed-shop or academic environment was assumed for the discussion to this point because of the nature of the research facilities available. Nothing appears to prohibit extending the presented ideas to public network situations provided that the assumptions in section 1.4 hold: a minority of traffic requires security guarantees, and the number of subversive individuals is small and they are interested in a small subset of total traffic. Such assumptions do not hold in financial or military environments [Padlipsky et al. 79, Simmons 79, Faught 81, Jones & Nelson 81].
7. Summary and Conclusions

This work has outlined some of the problems which may arise when attempting to provide end-to-end guarantees on the services of a CBMS. The conflicts arise from the desire to protect messages, following a design model which separates the message transport mechanism from the security measures and message preparation and manipulation, the basic one-way nature of messages, and attempting to provide the functionality commonly expected from such a system. The problems were: preventing reintroduction of a previously recorded (legitimate) messages, the inability to always obtain a digital signature after receiving a message if the sender has changed his secret key (using a public-key cryptosystem), the exclusion of a certain digital signature technique whether or not message retraction is allowed, providing end-to-end encryption for a message to be delivered to multiple destinations by the MTS, and the inability to provide guaranteed delivery confirmation by end-to-end methods or via the MTS. Cryptographic sealing techniques were shown to solve the fourth problem. In the case of assuring message time integrity, a temporal value was given and justified for the trade-off between database management costs and the probability of rejecting legitimate messages.

Other objectives given in section 1.2 have been realized: a CBMS model was described; an implementation following this model was outlined and evaluated; a general discussion of cryptosystems, security threats, key management, and placement of encryption protocols was given; and a model for user security was outlined. The preceding section discussed
possible enhancements to the implemented CBMS, the use of "dummy" messages to prevent traffic analysis, and the applicability of the ideas presented to a broader range of network situations.

Further research work in the areas addressed by this thesis in order of complexity include:

(a) a more precise definition of how a service provider fits into the CBMS model (for example, how does a UA or MTA communicate with it [IFIP 81b], by messages or some real-time means?);

(b) complete implementation of a CBMS according to the models given, including the use of an authentication server, directory server, and end-to-end techniques in the user agent program for message protection and delivery confirmation;

(c) extension of the message system onto a network with multiple hosts and a larger, more diverse user community;

(d) definition, justification, and adoption of format standards for message contents and the message envelope;

(e) definition and verification of a minimal set or kernel of operations for the top level of the MTS.
8. List of References

[Birrell et al 82]

[Booth 81]

[CCITT 80]

[Cheriton 79]

[Cheriton 80a]

[Cheriton 80b]

[Crocker et al. 79]

[Denning 79]

[Denning & Sacco 81]

[Deutsch 81]
[Diffie & Hellman 76]

[Diffie & Hellman 79]

[Dickson 80]

[Faught 81]

[Gait 77]

[Gifford 82]

[Hellman 80]

[IFIP 81a]

[IFIP 81b]

[Jones & Nelson 81]

[Kent 81]
[Lampson 73]
Lampson, B. W. "A Note on the Confinement Problem". CACM Vol. 16, No. 10 (October 1973), pg. 613-615.

[Landweber & Solomon 82]

[Lempell 79]

[Lockhart 79]

[McCauley & Drongowski 79]

[Merkle & Hellman 78]

[Michelman 79]

[NBS 75]

[NBS 77]

[NBS 81]
[Needham & Schroeder 78]  

[Neufeld 82]  
Neufeld, G. W. "EAN - The Design of a Distributed Message System". Computer Science Department, University of British Columbia, Vancouver, B.C., 1982.

[Padlipsky et al. 79]  

[Popek & Kline 79]  

[Price 81]  

[Rivest et al 78]  

[Rushby 81]  

[Saltzer et al. 81]  

[Schicker 79]  

[Schicker 81]  
[Simmons 79]

[Vittal 81a]

[Vittal 81b]

[Zimmermann 80]
Appendix A: DES Implementation

The following is a listing of function subroutines written in the language Zed [Cheriton 80b] which implement the DES algorithm [NBS 77] on the Verex Operating System [Cheriton 79, Lockhart 79]. Zed is a variant of the BCPL, B, and C languages and is used almost exclusively for software development under Verex.

Zed has been described as a "high-level assembly language". The language is machine oriented. It provides primitive data typing and checking. Data types are word-oriented. Structured programming control constructs exist. There are no I/O operations defined and run-time support is minimal; Zed programs utilize an "environment library" of standard or host-specific routines. A program consists of one or more modules, where a module is a function or data definition.

All input after a '\\' character to the end of a line is considered a comment in Zed.

The functions to perform the DES algorithm exist as relocatable modules in a subroutine library. They are linked in with a user program during the load operation. Normally the function '.DES_box' is the only one that would be invoked directly in an application program or function. The remaining functions are invoked by '.DES_box'. The global variables used are also included in the listing.

'.Alloc_vec' is a system library function which allocates a portion of dynamic memory and returns an address to it; it allocates a dynamic vector. The number of elements in the
To encrypt the information in vector 'plaintext' under DES using vector 'key' yielding vector 'ciphertext' (all three vectors having 64 elements, each element either 0 or 1 in value) within a Zed program, the following statement could be used:

`.DES_box( plaintext, key, ciphertext, .TRUE );`

Similarly, for decryption the following could be used:

`.DES_box( ciphertext, key, plaintext, .FALSE );`
\textbf{DES\_box( in[], key[], out[], encrypt )}
\begin{verbatim}
\textbf{Using DES algorithm, operate on vector "in" using "key"
yielding "out". "in", "key", and "out" are arrays of 64
elements (indexed 0-63), each element either 0 or 1.
If "encrypt" is .TRUE, encrypt information in "in".
If "encrypt" is .FALSE, decrypt information in "in".}
extrn .DES\_ip[], .DES\_ip\_[], .DES\_pc\_1[], .DES\_pc\_2[];
extrn .DES\_p[], .DES\_e[], .DES\_s[][];
extrn .DES\_left\_shifts[], .DES\_right\_shifts[];
extrn .DES\_k[], .DES\_k\_n[], .DES\_l[];
extrn .DES\_new\_r[], .DES\_r[], .DES\_work\_area[];
unsigned iter, s, t[];

.DESC\_initialize();

.DESC\_transpose( in, .DES\_ip, 64, .DES\_work\_area );
.DESC\_copy( .DES\_1, 0, .DES\_work\_area, 0, 32 );
.DESC\_copy( .DES\_r, 0, .DES\_work\_area, 32, 32 );
.DESC\_transpose( key, .DES\_pc\_1, 56, .DES\_k );

for( iter = 0; iter < 16; ++iter )
{
\begin{verbatim}
\textbf{generate new key according to key schedule}

if( encrypt )
    .DESC\_shift\_key\_left( .DES\_k, .DES\_left\_shifts[iter] );
else
    .DESC\_shift\_key\_right( .DES\_k, .DES\_right\_shifts[iter] );
.DESC\_transpose( .DES\_k, .DES\_pc\_2, 48, .DES\_k\_n );

\textbf{perform f(R,K)}

.DESC\_transpose( .DES\_r, .DES\_e, 48, .DES\_work\_area );
.DESC\_xor( .DES\_work\_area, .DES\_k\_n, 48 );
for( s = 0; s < 8; ++s )
    .DESC\_s\_box( .DES\_work\_area, s, .DES\_s[s] );
.DESC\_transpose( .DES\_work\_area, .DES\_p, 32, .DES\_new\_r );

\textbf{compute new L and R}

.DESC\_xor( .DES\_new\_r, .DES\_l, 32 );
t = .DES\_l;
.DESC\_l = .DES\_r;
.DESC\_r = .DES\_new\_r;
.DESC\_new\_r = t;
\end{verbatim}

.DESC\_copy( .DES\_work\_area, 0, .DES\_r, 0, 32 );
.DESC\_copy( .DES\_work\_area, 32, .DES\_l, 0, 32 );
.DESC\_transpose( .DES\_work\_area, .DES\_ip\_, 64, out );
\end{verbatim}
\end{verbatim}
.DES_copy( dst[], dst_index, src[], src_index, number )
\ copy elements src[src_index] ... src[src_index+number-1]
\ to dst[dst_index] ... dst[dst_index+number-1]
{
  unsigned cnt;
  for( cnt = 0; cnt++ < number; )
    dst[dst_index++] = src[src_index++];
}

.DESE_initialize()
\ If the work areas for the DES algorithm have not been allocated,
\ do so and set .DES_initialized to reflect this fact.
{
  extrn .DES_initialized, .DES_k[], .DES_k_n[], .DES_1[];
  extrn .DES_new_r[], .DES_r[], .DES_work_area[];
  if( !.DES_initialized )
  { 
    .DES_k = .Alloc_vec( 55, 1, 0 );
    .DES_k_n = .Alloc_vec( 47, 1, 0 );
    .DES_1 = .Alloc_vec( 31, 1, 0 );
    .DES_new_r = .Alloc_vec( 31, 1, 0 );
    .DES_r = .Alloc_vec( 31, 1, 0 );
    .DES_work_area = .Alloc_vec( 63, 1, 0 );
    .DES_initialized = .TRUE;
  }
}

.DESE_s_box( work_area[], s_number, substitution[] )
\ perform substitution function on bits in "work_area" and place result
\ back in "work_area".  "substitution" is the vector containing bits to
\ be substituted.
{
  unsigned index, row, col, iter, s;
  index = 6 * s_number; \ start of 6-bit group
  row = work_area[index++]<<1; \ 1st bit - high order bit of row #
  for( col = iter = 0; iter++ < 4; ) \ next 4 bits - column #
    col = (col<<1) + work_area[index++];
  row += work_area[index]; \ 6th bit - low order bit of row #
  s = substitution[16*row+col]; \ get substitution matrix entr
  index = 4 * s_number + 3; \ substitute back 4 bits
  for( iter = 0; iter++ < 4; ) \ convert # to 4 bits
  { 
    work_area[index--] = s & 1;
    s >>= 1;
  }
}
.DES_shift_key_left( key[], number_of_shifts )
{
    unsigned src_ndx, dst_ndx, t, i;

    for( i = 0; i++ < number_of_shifts; )
    {
        \ shift leftmost 28 bits
        t = key[0];
        for( src_ndx = dst_ndx = 0; dst_ndx < 27; )
            key[dst_ndx++] = key[++src_ndx];
        key[27] = t;

        \ shift rightmost 28 bits
        t = key[28];
        for( src_ndx = dst_ndx = 28; dst_ndx < 55; )
            key[dst_ndx++] = key[++src_ndx];
        key[55] = t;
    }
}

.DES_shift_key_right( key[], number_of_shifts )
{
    unsigned src_ndx, dst_ndx, t, i;

    for( i = 0; i++ < number_of_shifts; )
    {
        \ shift leftmost 28 bits
        t = key[27];
        for( src_ndx = dst_ndx = 27; dst_ndx > 0; )
            key[dst_ndx--] = key[--src_ndx];
        key[0] = t;

        \ shift rightmost 28 bits
        t = key[55];
        for( src_ndx = dst_ndx = 55; dst_ndx > 28; )
            key[dst_ndx--] = key[--src_ndx];
        key[28] = t;
    }
}
.DES_transpose( in[], permutation[], number, out[] )
\ Permute the elements of vector "in" according to "permutation"
\ and place the result in "out". "permutation" and "out" are
\ indexed from 0 to "number"-1.
{
  unsigned i;

  for( i = 0; i < number; ++i )
    out[i] = in[permutation[i]];
}

.DES_xor( x[], y[], number )
\ XOR corresponding elements of vectors x and y and place the
\ result in x.
{
  unsigned i;

  for( i = 0; i < number; ++i )
    x[i] 0 y[i];
}

\ Definition of global externals used for holding
\ work vectors of DES algorithm

.DES_k[];
.DES_k_n[];
.DES_1[];
.DES_new_r[];
.DES_r[];
.DES_work_area[];

\ Global flag indicating whether above work
\ vectors have been allocated memory: initial
\ value FALSE.

.DES_initialized: .FALSE;

\ The following vectors represent the permutations used in the
\ DES algorithm. Note that each element is one less in value
\ than given in the DES document. This is because Zed indexes
\ vectors starting with the 0'th element.

\ DES E-bit selection function

.DES_e[47]:
 31, 0, 1, 2, 3, 4, 3, 4, 5, 6, 7, 8, 7, 8, 9, 10, 11, 12, 11, 12, 13, 14, 15, 16, 15, 16, 17, 18, 19, 20,
19, 20, 21, 22, 23, 24,
23, 24, 25, 26, 27, 28,
27, 28, 29, 30, 31, 0;

\ DES initial permutation IP

\[\text{DES}_{ip}[63]: 57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7, 56, 48, 40, 32, 24, 16, 8, 0, 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6;\]

\ DES initial permutation (IP) inverse

\[\text{DES}_{ip}[63]: 39, 7, 47, 15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25, 32, 0, 40, 8, 48, 16, 56, 24;\]

\ DES permutation function P

\[\text{DES}_p[31]: 15, 6, 19, 20, 28, 11, 27, 16, 0, 14, 22, 25, 4, 17, 30, 9, 1, 7, 23, 13, 31, 26, 2, 8, 18, 12, 29, 5, 21, 10, 3, 24;\]

\ DES permuted choice function PC-1

\[\text{DES}_{pc_1}[55]: 56, 48, 40, 32, 24, 16, 8, 0, 57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35, 62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 60, 52, 44, 36, 28, 20, 12, 4, 27, 19, 11, 3;\]

\ DES permuted choice function PC-2

\[\text{DES}_{pc_2}[47]: 13, 16, 10, 23, 0, 4, 2, 27, 14, 5, 20, 9, 22, 18, 11, 3, 25, 7, 15, 6, 26, 19, 12, 1, 40, 51, 30, 36, 46, 54;\]
\ DES left shifts for key schedule
\ Note that in Zed, vectors are index starting with the 0'th element

\ DES_left_shifts[15]:
1,
1,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
1;

\ DES_right_shifts[15]:
0,
1,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
2,
1;

\ DES substitution box representations
\ Note that in Zed, vectors are index starting with the 0'th element

\ DES_s[7][63]:

\ S1
[ 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 ];
\ S2 
[ 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10, 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 ]

\ S3 
[ 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 ]

\ S4 
[ 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 ]

\ S5 
[ 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 ]

\ S6 
[ 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 ]

\ S7 
[ 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 ]

\ S8 
[ 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 ]