Security of Software Packages and Telecommunication Networks

Constantine Nicholas Salamis

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE

Presented in Partial Fulfilment of the Requirements for the Degree of Master of Computer Science at Concordia University
Montreal, Quebec, Canada

March 1983

SECURITY OF SOFTWARE PACKAGES AND TELECOMMUNICATION NETWORKS

CONSTANTINE NICHOLAS SALAMIS

ABSTRACT

The demand for more complex software to drive the current and future generations of computers is nonabating. The need for transmission of highly classified data is also on the increase.

This thesis brings to light the need to secure valuable software packages, important application programs, as well as the necessity to encrypt transmitted data.

Several methods, some old and some new, for software and data encryption are discussed and critically evaluated. Eventhough these methods cannot provide one hundred percent security from outside intrusion and theft; they are proposed with the intention of provoking one's thoughts on the subject and enabling the discovery of new ways of making it more difficult for an intruder to 'crack the safe'.

To my wife

Corinne

ACKNOWLEDGMENTS

To Dr. Alagar I wish to express my appreciation for his guidance, enlightenment and the many hours he spent helping me do the research on this thesis. His knowledge and insight helped provoke my thinking and understanding of the subject matter of this research topic.

My gratitude also extends to certain persons at my place of work. To Mr. George Sekely, Director of Information Systems at Canadian Pacific, and to my immediate supervisor Mr. Guy Vaillancourt for permitting me to use the Company's computer for executing the few programs needed for the analysis of the different areas of this thesis. My thanks also extends to Mr. George Simon, Operations Director also of Canadian Pacific, who made the printing of this thesis so easy for me by permitting me to use, and informing me of the many capabilities of our Xerox 9700 on-line printer. The list of my Company personnel who helped me would certainly not be complete without the mention of Mr. Nick Kuziomko who went out of his way to set up the complex procedures that made the printing of this thesis possible. Mention must also be made of Mr. James Rockwell who made sure there was a terminal available for me for the data entry. I thank them all very much.

A very special and warm thank you goes to my wife Corinne for her patience, understanding, and most of all, for the kind encouragement she gave me during the few moments of despair. Thank you Corinne.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

"Stolen waters are sweet, and bread eaten in secret is pleasant."[1]

To acquire that which does not rightfully belong to you, is certainly not a product of today's society. Stealing a loaf of bread to satisfy the pangs of hunger, happened many centuries ago. Man has devised many protective schemes to ward off the culprits' attempts to steal and thus protect his property. Numerous penalties proportional to the severity of the crime committed also have been devised and are administered in the courts once the wrong doers have been caught and convicted. The different types of items that are stollen, vary immensely. The common denominator is value. If an item is scarce and is in demand, it stands the chance of falling into the wrong hands at some time. Therefore, we may be at ease to know that it is not the advent of the computer that has brought on the urge to steal. Aside from any hardware 'misplacements', software logic and data suffer the same malady that befalls other items of value left unprotected.

1. The Holy Bible. Book of Proverbs, Chapter 9, Verse 17.

Mr. Donn Parker - consultant, expressing his views on protection in data processing, states that, "Safeguards introduced into a system against intentional attacks, work also to prevent accidental loss of data. While, safeguards built only to guard against accidental loss of data, usually do not work to fend off intentional attacks. Data processing deals with an entirely new environment. Electronic money is not cash in a drawer any more. Data processing goes on out of human view. The swift and quiet magnetization of ferite cores is all that it takes to commit any one of the below mentioned offenses. We cannot see and feel it. The wrong doing can occur right in front of us, but we cannot actually witness it. A computer is like the Trojan horse, huge and impressive looking on the outside, but with the ability to be potentially destructive on the inside. The world of data processing and everyone and everything it encompasses must learn to cope with such offenses as fraud, theft, larceny, embezzlement, sabotage, espionage, extortion and conspiracy."[1]

Just as there is no termination date for the Industrial Revolution, our present information explosion seems not to be abating in its momentum. We are an information hungry society, always wanting more to quench our unsatiated thirst for more knowledge.

The science of computing is very vast and

---

1. Mr. D. Parker - consultant - on a video tape of a DPMA Conference.

detailed. One can choose different parts of this realm as the nucleus with all the related items fanning out as spokes of the wheel of Computer Science, and still there will be someone who will choose another item stating that it is more important, and that it only should occupy the prestigious position in the nucleus. Possibly there are many nuclei. To indulge in describing the security necessary to protect each component of the computer as well as all its related activities, would be extremely lengthy. In this thesis we address ourselves to the security of software, static data[1], as well as secure telecommunication networks. We attempted to introduce and discuss some theories and suggestions on how best to protect this valuable information.

1.1 Security of Software Packages

The first area that this thesis addresses, is that of software package security.

The word 'software' seems to have had to be under some pressure to come into existence, in that it melds well with the 'hardware' of the computer itself. Software ranges from the simplest of programs written with a few instructions, to the vast and most complex packages like those of the Operating Systems. We do not wish to give the impression that these vast Operating Systems are always more important than those small, seemingly insignificant programs

1. Meaning of static data is non-transmitted data as opposed to dynamic data that is transmitted from one point to another.

that could be the logic executing highly classified formulae or the interface between persons entering their passwords from a terminal and highly classified data residing in auxiliary storage. It is this program that will make the final decision whether to grant access or refuse it. Many man-years have gone into designing, writing, testing and finally implementing these software packages. The enormous cost to produce the end product must certainly not be forgotten. When these programs fall into the hands of unauthorized persons like a competitor, it could possibly spell out the downfall of the company that was lucky enough to originate such technical and valuable products.

It is of utmost importance therefore, to take steps to secure software that is of great value. Development costs as well as the length of time to produce the finished package that is ready for marketing are almost always the reasons for small inefficient companies to want to steal them.

The software house industry has flourished in the past couple of decades. Their software packages are not to be confused with the supplier Operating System packages that usually accompany the hardware installation. Vast and highly complex operating systems and application programs belonging to the software houses and hardware suppliers are items of great value. Not only would their competitors like to get their hands on these software packages, but also individuals that are new vendors and who wish to expedite their entry into the market with minimum investment of resources. They merely wish not to take the time, trouble, and expense to develop software packages or application programs of their own for sale or lease to others. At issue is the protection of software from theft and the establishment of unquestionable proprietary rights on behalf of the creators of software products. Richard DeMillo {21} refers to proprietary software as a computer program that has a wide potential use and also reflects a better than average level of industry and/or computer expertise which is to be sold or leased at a fraction of the cost it would take for any one computer installation to program themselves.

These software products leave the premises of the supplier or software house. Consequently, the creators cannot police the use, or misuse, of their product by their customers. There is also the problem of "fixes". These are corrections to errors in the logic that have sprung up during the course of initial program usage by the customer. The creators just did not have the insight, or it may have been a legitimate piece of logic temporarily included in order to trap certain situations that do arise from time to time. Because of wide-spread distribution of their products, lack of personnel, and other related reasons, the software supplier will usually send the corrections to the customer for implementation by the latter's staff. Consequently, there is some 'uncovering' of the logic by the customer. The memory dumps do reveal the myriads of instructions that make up the operating system. A person with knowledge and plenty of time, can in effect decipher the assembler code, pocket it and nonchalantly walk out of the front door with it.

Richard DeMillo {21} is very concerned that, with the lack of adequate protection mechanisms, and with current methods of transacting business for the leasing of software packages, there is an increased risk associated with the proprietary software market. What can be done? Is the vendor at the mercy of the customer? A mechanism must be found to protect the property of software vendors if any sustained growth in the commercial software industry is to be maintained.

There is the possibility of improved legislation or the development of new technology to ensure that proprietary rights to computer software can be established and maintained. Some basic current legal protection consists of patent laws, copyright, trade secrets, licensing, and built-in hardware safe guards. Each is briefly discussed below.

Patent laws have the draw back in that they cannot be used to protect ideas. They can only be used to protect the results emanating from these ideas.

Copyright is the exclusive legal right to reproduce but is limited only to the physical aspects of a program. One cannot cover algorithms and data structures on which a program is based. There seems to be protection only from blatant reproduction.

Trade secrets are used to hide features and concepts embodied in the packages or any combination of algorithms and data structures that address the specific goals or problems. Laws concerning trade secrets and their divulgence thereof, vary from state to state and are loosely defined at the Federal level. {21}.

Licensing is a contract that binds a customer to certain obligations not to divulge certain parts of software packages. These contracts also permit him to receive from the supplier or vendor, highly classified modules he requires for his day-to-day operation.

Built-in hardware safeguards are being used more and more by hardware suppliers who also may produce the software operating systems. The manufacturers hard wire highly sophisticated logic. These manufacturers circumvent laws that could prevent them from doing this by merely constructing a new model of their computer line, along with a a new version of their software that is in effect mainframe dependent. The advantages to this scheme seem to outweigh the drawbacks. Not being able to run a particular supplier's software due to its hardware dependence, puts a damper on stealing it. Hard coding instructions increases processing speed. Among the major drawbacks to hard wired logic, is that it is costly to alter.

The ability to change one's own hardware to the point where a competitor's software package cannot be implemented, is a significant step taken in thwarting a customer's desire to choose the hardware from one supplier, and the software which may be 'hot', from another.

1.2 Security of Data by Encryption

The second area that this thesis addresses, is the encryption of data.

The data referred to here in this section, is static data, as well as data that is transmitted from one point to another. Besides encrypting the classified pieces of information a software package or application program uses, we also involved ourselves in analyzing methods in security of networks and data transmission.

The pieces of hardware that we discuss here are, the terminal, the modem, the carrier, and the network. A brief discussion of each appears below:

The Terminal:

This is a piece of hardware that interfaces between the user and the telecommunication media. It may be an intelligent or dumb terminal. For our purposes, we will think of it as a black box that permits us to enter data, prints this data on paper mounted in a similar fashion as a normal typewriter, or displays it on a screen of a cathode ray tube, or CRT. Any information received by this black box is displayed in the same manner.

The Modem:

This is the next piece of hardware encountered. A modem primarily modulates and demodulates the electrical signal in order that it may be used by the terminal, a received transmission, or prepares the signal to be handled by the carrier just prior to sending.

The Carrier:

The carrier is the medium used for carrying the actual signal. This medium may be of different types. A message may be 'carried' by these different types as it travels from sender to receiver. Following is a brief description of the most common forms of telecommunication carriers.

Wire:

A solid piece of metal that is used to directly connect two terminals. The electrical pulses travel along this solid piece of cable.

Micro Wave:

The medium here is electromagnetic waves sent through the atmosphere from one microwave tower to another located a few miles apart.

Laser Beam:

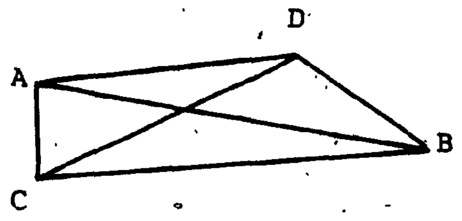
The laser beam is the latest breakthrough in signal conveyance. In this case the signal rides on a high intensity beam of light.

At the receiving end we encounter another modem. This time, as mentioned before, the signal is demodulated, preparing it for use by the terminal. Finally, the signal is recognized by the receiver's terminal and is displayed in legible form.

The Network:

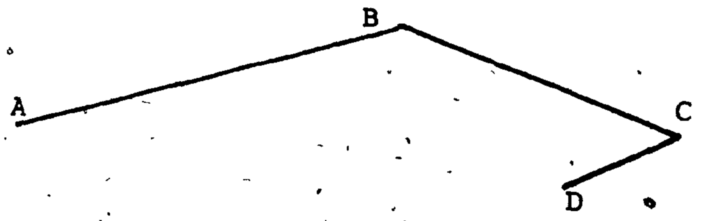
In the previous sections we discussed only two terminals with a carrier between them. We must now think of many users, each with his own terminal, wanting to communicate with every other user in this family of terminal sites. For this to be possible, each terminal must have a direct or indirect path to the terminal(s) it wishes to communicate. We will henceforth refer to this family of terminals and the interconnecting carriers as a communications network.

In Example 1.1 each terminal is linked directly to all the other terminals.



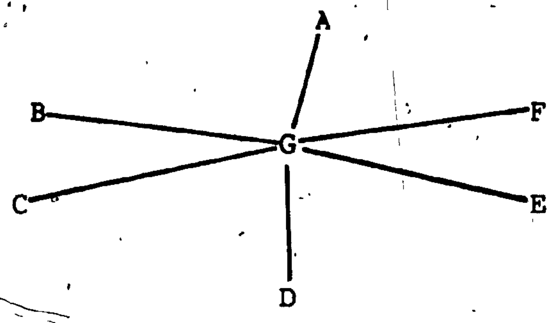
EXAMPLE 1.1

In Example 1.2 each terminal may communicate with every other terminal, just as in Example 1.1, but in an indirect manner.



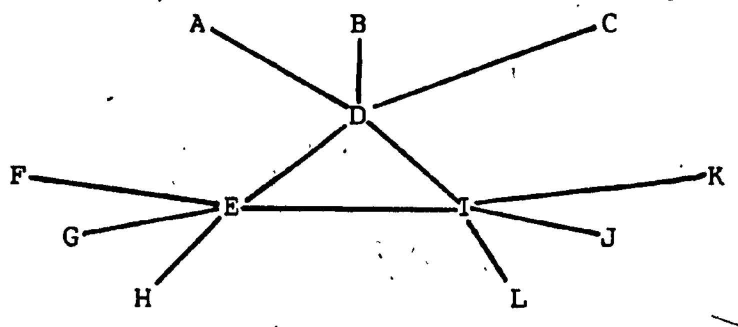
EXAMPLE 1.2

In example 1.3 every node, or terminal site, is connected to every other node but again indirectly through the special node G that will be discussed in a later chapter. Suffice it to say for the present, node G is a message switching computer that directs traffic to its proper destination.



EXAMPLE 1.3

We can look upon Example 1.4 as three separate networks. Each node is connected to its special message switching node. These special nodes are themselves interconnected providing for inter-network transmission.



EXAMPLE 1.4

There is a myriad of different network topologies. Terminal sites may be in:

1. different rooms in the same building
2. different buildings in the same city
3. different cities of the same country

4. different countries on the planet Earth
5. different heavenly bodies of the same planetary system, as was the case between the earth and the moon when the latter was visited by man.

1.3 The Security Problem

A person, or a device, called the originator, wishes to send a message to another person(s) or device(s) called the receiver. The inherent problem is that stringent methods must be implemented to prevent this message from either not being received by an unauthorized person(s), or device(s), or that if the carrier is tapped and the message is siphoned off, its text is so changed that its context is never understood. In simple terms, we want to make sure that a third party is not listening in on the conversation going on between originator and receiver.

If the data is of no intrinsic value to others, then there is no attempt to hide the meaning of this data in any way, shape, or form. We in effect do not care if someone else taps our communication link and is able to read and understand the text being sent.

If on the other hand the message being transmitted has great value and is consequently regarded as highly classified material, then we certainly care if a third party intercepts our message. We wish to encrypt our message so that it will not be understood by the impostor.

To make a telecommunications network impregnable by outside unauthorized forces, involves making key components secure. Some are briefly discussed.

1. Signing on to a Network

The family of users wants to make absolutely sure that unwanted guests are caught when attempting to use their communication network. Hardware and software techniques to ward off potential eavesdroppers help ease the uncomfortable feeling the members of the network family may have knowing that someone may be listening in. The terminal may be hardwired in such a way as to accept certain codes typed in or, certain data read from magnetic cards the size of a credit card inserted into a password-card reader attached to the terminal. There may be a lock on the terminal accepting only a certain key. Without the insertion and turning of the correct key, the terminal remains inoperable.

Software wise, the terminal may have a buffer wherein a small program may reside whose logic checks for the correct secret code typed in by the user.

2. The Protocol

A protocol could be thought of as the 'handshaking' that takes place between the originator of a message and the receiver. A third entity is quite often involved, called an authentication server, or as David L. Chum {4} calls them, "the mixes". These mixes are usually none other than message switching computers. In Chapter 5 we will discuss two authentication methods of protocol, the Conventional Method, and the Public Key Method. The two papers, R.M. Needham, M.D. Schroeder {14} and G.J. Popek, C.S. Klein {16} both deal with this important part of secure communication that takes place just prior to the actual transmission of the text of a message.

3. Encryption and Transmission of Text

Many algorithms have been devised to 'scramble' the characters as well as the bits of the characters making up the text of a message. John B. Kam and George I. Davida {23} involve themselves with algorithms to interchange the position of the bits making up the characters using certain permutations. The receiver, knowing the permutations used, would then appply an unscrambling algorithm to decipher the encoded text. R. L. Rievst, A. Shamir, and L. Adleman {17}, encrypt a message by representing it as a number M, raising M to a publically specified power e, and then taking the remainder when the result is divided by the publically specified product, n. Here n is the product of two large secret prime numbers p and q.

4. Digitalized Signatures

The network system must have complete certification of all messages. The receiver of a message just received, must be completely assured as to the authenticity of the originator. Otherwise, suspicion may easily ensue. It is of the utmost importance the receiver be at ease in this respect, otherwise the classified message will not be believed by the recipient. R.L. Rivest et al {17}, besides text encryption, also touch on the area of digitalized signatures, as does Michael O. Rabin {27}. Analysis of these and other methods for digitalized signatures are discussed in Chapter 6.

Chapter 2 of this thesis will endeavour to describe some means of protecting software and making it as secure as possible.

Chapter 3 looks at the telecommunication network and describes how a minimum cost network may be designed.

Chapter 4 looks at some ways and means of protecting a network from unauthorized persons gaining access to the system through the use of a terminal. Some strict logon procedures are discussed.

Chapter 5 details what actually goes on during the handshaking period between the message originator, mixes when they exist and the receiver, just prior to transmission of the message.

Chapter 6 deals with some methods that might be employed in encrypting the text of messages for secure communication between two entities.

Chapter 7 is the conclusion of the thesis.

CHAPTER 2

SOFTWARE PROTECTION AND SECURITY

2.1 Preamble

In this chapter we endeavour to show how software may be protected and made secure from persons who wish to steal its contents for use by themselves or resale to others. The two main sections of this chapter may be labelled as, indirect and direct protection.

2.2 Indirect Protection

By indirect we mean the laws that exist to protect the originators of such packages from theft. Under this heading we can find copyright, patent, and trade secret laws that are supposed to deter the potential thief. Laws against hybrid piracy, wherein the core of the logic is stolen and is embellished with the thief's own logic, are among the laws mentioned above. Unfortunately these deterrents are not very effective. In our opinion, what seems to work more satisfactorily are the direct methods of software protection.

Some direct methods are now discussed.

2.3 Direct Protection

A program in a certain state can be said to be self-encrypting. The instruction code of a software package could exist in several formats. Programs written in high level language resemble very closely to plain English text. The originators of said languages wanted not only to make writting programs easy, but also wanted to facilitate the reading and changing of these programs by others who did not originally write them. In introducing this easy to read text, a program in source code can easily be read and quite well understood even by a person who is not in the computing field. Programs in source code are vulnerable. If the same logic is to be scanned after it has become an object module, it would be extremely difficult to translate it back to its source code, and then to English. Therefore, one might say that programs in object or load module code are self-encrypting. These modules, as well as application programs in source code, reside on secondary storage. Magnetic tapes, magnetic disks, and magnetic drums are the most popular auxiliary storage devices. Bubble memory is still in its infancy stage. On these devices as well as in main memory, instructions and any static data look like, and can be assumed to be data. A dump of the auxiliary storage or of main memory, would reveal strings of numbers and letters. In reality, this data is merely a string of zero and one bits. In general, in many systems, differentiating between instructions and data is quite difficult. Consequently, the problem boils down to the encryption of data, where data, as described above, includes instructions, data and any other coding that goes into making the package complete and workable.

2.3.1 Method A - Mathematical Text Manipulation

The first method of text encryption analyzed was encryption by mathematical text manipulation.

This method is outlined in detail in Chapter 6. The analysis is concerned with encrypting text prior to transmitting it over insecure communication lines. This method is not peculiar only to transmitted data, for it may also be used in encrypting static data just prior to storing it in auxiliary storage. This may be carried a step further, in that, if it is required to record very detailed documentation on valuable software, this method may also be applied without any problems.

R. L. Rivest et al {17}, explain, "The text to be encrypted is represented as a number M, raising M to a publicly specified power e, and then taking the remainder when the result is divided by the publicly specified product n, of two large prime numbers p and q".

2.3.2 Method B - Superimposed Coding

A second method studied was that of superimposed coding. The case in point here is the use of superimposed coding in securing a decision table. We may find a very good example in the computer programs run by Revenue Canada. These tables describe certain demographic attributes of an individual. For example, items such as:

SEX:	Male or Female
MARITIAL STATUS:	Married or Single
	Widowed or Divorced
Citizenship:	Canadian or Non-Canadian
Houseing:	Owns a home or rents
Salary Range:	10K to 11.9K
	12K to 13.9K
	.
	.
	etc.

EXAMPLE 2.1

These 'n' attributes can be represented as:

$x_1, x_2, x_3, \ldots\ldots, x_n$.

The presence of an attribute in this table may be denoted as $x_i = 1$, while its absence is represented as $x_i = 0$. This is of course much too obvious. To encrypt this string of zero and one bits, each attribute is associated with two k-bit words, w and w'. Certain m bits, where m << k, are set on in w_i if the attribute is present. If the attribute is absent, then a certain pattern of m bits is turned on in w_i'. An example follows.

m = 2, k = 5	Positive Response (w)	Attribute x_i	Negative Response (w')
	00000	$x_1 = 0$	10100
	10010	$x_2 = 1$	00000
	10001	$x_3 = 1$	00000
	00000	$x_4 = 0$	00011

EXAMPLE 2.2

The inclusive OR is taken of the positive and negative words separately. In Example 2.2, the w's result would be 10011, while that of the w' 's would be 10111.

We may now associate a predicate $P(x_1, x_2, \ldots. x_n)$ with a particular bit pattern in w and w'. An action(s) takes place as a result of the evaluated predicate. The particular predicate is either true or false. If true then either actions are initiated, or another predicate is interrogated. See Example 2.5.

Due to the nature of the inclusive OR, similar bit patterns in the result are inevitable. These particular bit patterns would represent different decision table results.

Positive Response (w)	Attribute x_i	Negative Response (w')
10010	$x_1 = 1$	00000
00000	$x_2 = 0$	10010
00000	$x_3 = 0$	00101
00011	$x_4 = 1$	00000
10011		10111

EXAMPLE 2.3

As may be seen, the results of Examples 2.2 and 2.3 are identical, eventhough the decision tables are totally different. These duplicates are better known as false drops. The frequency of these false drops occurring, are discussed by D.E. Knuth {25}.

We attempted to reduce the frequency of false drops for this problem. Since it is the nature of the inclusive OR operation as well as the position of the 0's and 1's in the different patterns that cause the duplicates, we set out to come up with some kind of pseudo operation that would render the results as unique as possible. If this method was to be chosen by someone for encrypting decision tables, the problem of false drops would have to be contended with, since duplicates persist.

The first attempt to reduce the number of false drops was with the use of a pseudo inclusive OR operation, The first step involves identifying the position of the '1' bits, or the 0 bits could have been considered instead, within the w's and the w' 's. The position number is established by counting from right to left. The bit pattern 001011 has '1' bits in positions 1, 2, and 4. The position numbers are then added for that particular w_i or w_i'. The above example would result in a total of 1 + 2 + 4 = 7. After the position summation is done for all the w_i's and w_i' 's, the respective results are then added together to produce two totals. A final step includes summing the indices of the words taking part in the OR'ing operation.

For brevity, only one side will be dealt with in this example. Only the w 's will be considered. The same analogy of course applies to the w' 's.

From prior analysis, it was found that best results would occur if not more than 1/3 of the total number of bits in a word, in our case, k, should be set to 1. In this analysis the value of k was chosen as 6, therefore within any one word, not more than 2 bits will be set on. The number of attributes, n, was taken as 10. The number 10 was chosen again for brevity's sake with no loss in theoretical value. The bit patterns describing each of the 10 attributes when their result is positive, are as follows:

$$\begin{array}{ll}
x_1 : & 110000 \\
x_2 : & 101000 \\
x_3 : & 100000 \\
x_4 : & 010000 \\
x_5 : & 100001 \\
x_6 : & 011000 \\
x_7 : & 010100 \\
x_8 : & 010010 \\
x_9 : & 001000 \\
x_{10} : & 001100
\end{array}$$

EXAMPLE 2.4

Some false drops were created as a result of inclusive OR'ing.

$$\begin{array}{llll}
\text{(a)} & \begin{array}{ll} x_1 : & 110000 \\ x_2 : & 101000 \\ x_7 : & 010100 \\ \hline & 111100 \end{array} & \text{(b)} & \begin{array}{ll} x_1 : & 110000 \\ x_7 : & 001100 \\ \hline & 111100 \end{array}
\end{array}$$

$$\begin{array}{llll}
\text{(c)} & \begin{array}{ll} x_3 : & 100000 \\ x_6 : & 011000 \\ x_7 : & 010100 \\ x_{10} : & 001100 \\ \hline & 111100 \end{array} & \text{(d)} & \begin{array}{ll} x_3 : & 100000 \\ x_7 : & 010100 \\ x_9 : & 001000 \\ \hline & 111100 \end{array}
\end{array}$$

In all these cases the result of the OR'ing is the same, that is, 111100. Applying the previously metioned steps, we obtain:

$$\text{(a)}\quad \begin{array}{rl} x_1 : 5 + 6 = & 11 \\ x_2 : 4 + 6 = & 10 \\ x_7 : 3 + 5 = & 8 \\ & \text{--} \\ & 29 \end{array} \qquad \text{(b)}\quad \begin{array}{rl} x_1 : 5 + 6 = & 11 \\ x_{10} : 3 + 4 = & 7 \\ & \text{--} \\ & 18 \end{array}$$

$$\text{(c)}\quad \begin{array}{rl} x_3 : 0 + 6 = & 6 \\ x_6 : 4 + 5 = & 9 \\ x_7 : 3 + 5 = & 8 \\ x_{10} : 3 + 4 = & 7 \\ & \text{--} \\ & 30 \end{array} \qquad \text{(d)}\quad \begin{array}{rl} x_3 : 0 + 6 = & 6 \\ x_7 : 3 + 5 = & 8 \\ x_9 : 0 + 4 = & 4 \\ & \text{--} \\ & 18 \end{array}$$

The next step entails adding the sum of the indices of the attributes that took part in the different summations and adding these totals to the respective totals obtained.

(a) $29 + (1 + 2 + 7) = 39$ (b) $18 + (1 + 10) = 29$

(c) $30 + (3 + 6 + 7 + 10) = 56$ (d) $18 + (3 + 7 + 9) = 37$

Eventhough false drops did not occur amongst these four examples, it was felt that the extra step of adding the sum of the indices to the sum of the bit positions would not really decrease the proportion of false drops to any significant level. Adding different numbers to totals already produced by addition, does not have any mathematical grounds for producing unique numbers. An analysis was done to determine the proportion of false drops. Instead of adding the position total, a tuple was formed. The first part of the tuple was the position total, while the second portion was the sum of the indices. In our four examples the tuples so formed were:

(a) (29,10) (b) (18,11)
(c) (30,26) (d) (18,19)

Of course tuples would also be calculated for the w' 's. Consequently, we are dealing with a two-paired tuple for different settings of the original decision table. These two pairs determine a predicate, and the actions to be taken. Below, (w,x) is the value of the tuple formed when the w's are operated upon, while (y,z) is the result when the operation is done on the w' 's. The respective instructions between THEN DO and END would be executed only when the values of the tuples was the same as the specific values represented by the letters a,b,c,d,e,f,g,h,......

```
IF P((w,x),(y,z)) = ((a,b),(c,d))
THEN DO;
        .
        .
        .
        END;
IF P((w,x),(y,z)) = ((e,f),(g,h))
THEN DO;
        .
        .
        .
        END;
```

EXAMPLE 2.5

If in the above example (w,x) (y,z) were evaluated (53,94) (6,4), then a certain set of instructions carrying out a specific decision, would be executed. Furthermore, if the variables w, x, y, and z emerged with other values, then another set of instructions would be executed denoting the requirement for another decision(s) to be taken.

Originally, 15 attributes were to be considered for this analysis. Unfortunately, due to computer time limitations, the sample size was trimmed slightly to a maximum of 14 attributes.

In our analysis, sample sizes were incremented by 1 from a minimum of 10 attributes to a maximum of 14. Each sample size was analyzed twice, once with both the respective w and w' words having the same bit configuration, and once with them having a different bit configuration. For instance, if x_5 was true, w_5 would have a bit configuration of 010010. If x_5 was false then during the first phase of the analysis w_5' had the <u>same</u> bit configuration as w_5, namely 010010. For the second phase of the analysis, w_5' would have a <u>different</u> bit configuration than its counterpart, w_5, say 111000.

The search for similar pairs of tuples was done as follows. The 1st and 2nd pair, the 1st and 3rd pair,... the first and the last pair. The following scan compared the

2nd and 3rd, the 2nd and 4th,..... until all possible combinations of comparisons were made.

Sample sizes of 1000 two-paired tuples were taken in order to determine if the proportion of false drops was consistent. Once each batch of 1000 was analyzed, the total number of items within each run was scanned as if it was a batch. The results are shown in Table 2.1

TABLE 2.1

ANALYSIS OF FALSE DROP OCCURRENCE

	SAME BIT PATTERN			DIFFERENT BIT PATTERN	
Items in Decision Table	Sample Size	No. of Dupl.	% of Sample	No. of Dupl.	% of Sample
10	1000	300	30.0	76	7.60
	23	0	0.0	0	0.00

	1023	300	29.33	76	7.43
11	1000	196	19.6	104	10.40
	1000	221	22.1	122	12.20
	47	0	0.0	0	0.00

	2047	924	45.14	514	25.11
12	1000	172	17.2	52	5.20
	1000	195	19.5	60	6.0
	1000	176	17.6	56	5.6
	1000	240	24.0	83	8.3
	95	0	0.0	0	0.0

	4095	3616	88.30	1252	30.57
13	1000	116	11.6	20	2.0
	1000	128	12.8	20	2.0
	1000	113	11.3	20	2.0
	1000	164	16.4	32	3.2
	1000	112	11.2	16	1.6
	1000	187	18.7	39	3.9
	1000	144	14.4	33	3.3

	1000	195	19.5	40	4.0
	191	9	0.05	0	0.0

	8191	10312	125.89	2872	35.06
14	1000	180	18.0	40	4.0
	1000	183	18.3	40	4.0
	1000	177	17.7	40	4.0
	1000	187	18.7	36	3.6
	1000	165	16.5	36	3.6
	1000	196	19.6	36	3.6
	1000	195	19.5	52	5.2
	1000	233	23.3	36	3.6
	1000	173	17.3	27	2.7
	1000	233	23.3	24	2.4
	1000	217	21.7	58	5.8
	1000	240	24.0	24	2.4
	1000	153	15.3	26	2.6
	1000	234	23.4	36	3.6
	1000	160	16.0	54	5.4
	1000	180	18.0	36	3.6
	383	16	0.04	4	1.04

	16383	30984	189.12	7160	43.70

Table 2.2 contains the results from the analysis.

TABLE 2.2

No. of Items in Decision Table	Avg. No. of Dupl. Per 1000 Comps. Bit Pattern in w & w' the Same	Avg. No. of Dupl. Per 1000 Comps. Bit Pattern in w & w' Different
10	300	76
11	209	113
12	196	63
13	145	28
14	194	38

From Table 2.2 it can be seen that false drops did occur. It was noted that the proportion of duplicates was less when the bit configuration is different for the x_i

attribute in w_i and w'_i. From Table 2.1, the proportion of false drops did not seem to be too constant from sample to sample. In sample sizes of 1000 comparisons, the number of duplicates when the w and w' bit patterns were the same, ranged from 112 to 300. When the bit patterns were different, the range was 16 to 122.

In order to further reduce the number of duplicates, the following changes were made:

(1) the number of bits per word was increased from the present 6,

(2) the position of the '1' bits was randomly generated,

(3) there was a minimum and maximum number of '1' bits per word that had to be met.

Table 2.3 indicates the different combinations that were analyzed. Table 2.4 depicts the results of the second attempt to reduce false drops.

TABLE 2.3

COMBINATIONS ANALYZED

Trial No.	Number of Words	Number of Bits/Word	Min. No. of '1' Bits	Max. No. of '1' Bits
1	10	10	5	7
2	10	15	8	10
3	10	20	10	14
4	14	10	5	7
5	14	15	8	10
6	14	20	10	14

TABLE 2.4

SUMMARY OF RESULTS FROM SUPERIMPOSED CODING

Trial No.	Batch No.	Sample Size	Number of Duplicates	Percent of Sample Size
1	all batches		NIL	
2	all batches		NIL	
3	all batches		NIL	
4	3	1000	4	0.4
	7	1000	8	0.8
	population	16383	376	2.3
5	all batches		NIL	NIL
	population	16383	64	0.4
6	1 to 16	1000 @	4 @	0.4 @
	population	16383	384	2.3

2.3.3 Concluding Remarks

The phenomenal decrease in the the number of duplicates between the 1st and 2nd attempts, may be attributed mainly to the random bit positioning as opposed to setting fixed patterns. Best results were obtained for a decision table size of 10. For a decision table size of 14 words, the results were not consistent. Taking all the samples together, and working on them as if they were one batch, the number of duplicates decreased from 376 to 64 with an increase of 10 to 15 bits per word, but increased from 64 to 384 when we increased the word length a further 5 bits, that is, to 20 bits per word. This is only an increase of 8 duplicates in a sample size of 16383. Not a significant amount. The proportion of duplicates per sample sizes of 1000 was much more consistent in the second attempt than in the first, having values of only 4 and 8, no matter how big the words. From our test results, we consider superimposed coding as a good method for encrypting software packages whenever they can be represented as a decision table. If this method is to be used, it is best to test a particular situation with different sized words and to choose the word size giving the least number of false drops.

2.3.4 Method C - Mathematical Message Manipulation

This method is outlined in detail in Chapter 6. The analysis is concerned with encrypting text prior to transmitting it over insecure telecommunication lines.

The text of a software package, as well as data, may be encrypted using this method just prior to storing it in auxiliary memory. This may be carried a step further, in that, if it is required to record on paper or on microfiche very detailed documentation of valuable software, this method may also be used without any problems. See {17} for details.

2.3.5 Method D - Substitution Permutation - 1

The following methods of bit scrambling may be used to encrypt stationary data, or data that will undergo transmission. These methods will be described later in this chapter.

The problem at hand is to scramble the bits of characters, or for that matter, characters themselves, in as complicated a manner as possible. We attempt to stump the intruder who wishes to decrypt the cipher text. Excluding luck, the more complicated the rearrangement, the longer it should take the impostor to identify what the clear text is. We wish that the intruder encounter maximum difficulty. The shuffling of the bits, or characters, should not have any cycles that can easily be unravelled by the outsider.

Reshuffling the bits in a string is equivalent to generating a permutation. One kind of interesting and hard to find permutation is related to placing non-attacking queens on a chess board.

Consider a chess board with a 4 X 4 matrix. On this chess board there exist 4 queens. As per usual, these queens may attack along any row, column, or diagonal. The object of the game is to place these queens in positions such that they cannot attack each other. Figures 2.1 and and 2.2 depict the only two solutions.

If we let P represent the first matrix, then the only other possible solution having the same constraints is the inverse of P, or P^{-1}.

$$P = \begin{array}{|c|c|c|c|}\hline & x & & \\ \hline & & & x \\ \hline x & & & \\ \hline & & x & \\ \hline\end{array} \qquad P^{-1} = \begin{array}{|c|c|c|c|}\hline & & x & \\ \hline x & & & \\ \hline & & & x \\ \hline & x & & \\ \hline\end{array}$$

FIGURE 2.1 FIGURE 2.2

Permutations may be set up describing the positions of the queens.

$$P = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \end{pmatrix} \qquad P^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix}$$

As for matrix P: the bit in position 1 ====> 2
" " " " 2 ====> 4
" " " " 3 ====> 1
" " " " 4 ====> 3

If a 4-bit string prior to permuting was 1010, using permutation P, it would become 1100. The chess board game may be played with a larger matrix as well as more queens. In practice, there are more than 4 bits involved. For a larger bit string, instead of using only one randomly chosen permutation from an n X n matrix, where n is the total number of bits to be scrambled, we could chop up the bit string into smaller unequal segments. For example, a string of 32 bits could have 4 unequal sections of 7, 14, 6, and 5 bits respectively. Consequently, different permutations would be selected for each section. In so doing, an intruder would have to figure out how many sections exist, the bit length of each section, and the permutation of each section.

We have not tested this method and recomend it for further study.

2.3.6 Method E - Substitution Permutation - 2

This method outlined by J.B. Kam and G. I. Davida {23}, is in effect a more complicated scrambling of the bits that represent the characters of a text. As with the previous methods, this substitution-permutation encryption method may be used without any modifications to encrypt static data, or software packages.

In this method the bits making up a set of characters pass through different stages wherein they are scrambled according to some logical pattern. Each stage is partitioned into groups. In turn, each group contains a number of 'boxes'. It is these boxes that contain the actual bits that make up the characters of the text to be encrypted.

Definitions- n:	k^t = number of I/O bits to be encrypted,
S/P:	Substitution-Permutation,
k:	number of I/O bits for each S/P box,
t:	number of S/P stages,
S_i :	$k^{(t-1)}$ S/P boxes per stage

In this analysis: no. of stages t = 3

no. of bits/box k = 3

no. of bits undergoing repositioning = $n = k^t = 3^3 = 27$

no. of boxes/stage = n/k = 27/3 = 9

The 1st. stage has $k^{STAGE\#-1}$ boxes in each group = 3^{1-1} = 1

The 2nd. stage has $k^{STAGE\#-1}$ boxes in each group = 3^{2-1} = 3

The 3rd. stage has $k^{STAGE\#-1}$ boxes in each group = 3^{3-1} = 9

The possibility of the internal structures of all S/P boxes and permutations becoming known to some outsider still exists. To further secure this design, provision is made for a choice of more than one permutation within any one S/P box. For example, the various permutations could be the results from the challenge of the queens on a chess board discussed previously or similar to the permutations depicted in figure 2.3.

A hidden key would be set up instructing which permutation is to be used within any one box at each stage. For example, if within any one box there was a choice of two permutations and it was understood that if the key bit indicated a '0', the first permutation would be used. If on the other hand the key bit was set to '1', the the second permutation would be used. A stage of nine boxes could have a key such as 110001010. We restricted ourselves to the use of one permutation per box, since multiple permutations are merely an extension to the basic one per box. The algorithm follows.

```
INTEGER STAGE#,GROUP-OFFSET,BIT#,BOX#,LAST-BOX#,LAST-BIT#,
     BIT-OFFSET;


  /* The following will connect the inputs of the      */
/* STAGE# th. stage to the outputs of the (STAGE#-1)  */
/* st. stage.                                          */
FOR STAGE#:=2 UNTIL t by 1 do;
/* The S/P boxes in the STAGE# th. stage are parti-    */
/* tioned into groups of $k^{STAGE\#-1}$ boxes each    */
    FOR GROUP-OFFSET =0 UNTIL ($k^{t-1}$ - $k^{STAGE\#-1}$)
                                   by $k^{STAGE\#-1}$ DO;
/* The 0th. input bits of the $k^{STAGE\#-1}$ S/P boxes in */
/* each group are connected to the first $k^{STAGE\#-1}$   */
/* output bits of the same group of the previous       */
/* stage, the ~~first input~~ bits of the $k^{STAGE\#-1}$ S/P */
/* boxes to the next $k^{STAGE\#-1}$ output bits of the group*/
/* of the previous stage..etc. as shown in FIGURE 2.3.*/
        FOR BIT#:=0 UNTIL (k-1) BY 1 DO;
            BIT-OFFSET:=BIT#*$k^{STAGE\#-1}$;
              FOR BOX#=0 BY 1 UNTIL ($k^{STAGE\#-1}$ - 1) DO;
```

```
                    PRESENT-BOX#:=GROUP-OFFSET+BOX#;
          LAST-BOX:=GROUP-OFFSET+((BOX#+BIT-OFFSET)/k);
                    LAST-BIT:=BOX-NO MOD (k);
  STAGE#(PRESENT-BOX,BIT#):=STAGE#-1(LAST-BOX#,LAST-BIT#);
                        END;
                    END;
                END;
            END;
```

3 STAGES, 9 BOXES/STAGE, 3 BITS/BOX, 27 BITS/STAGE

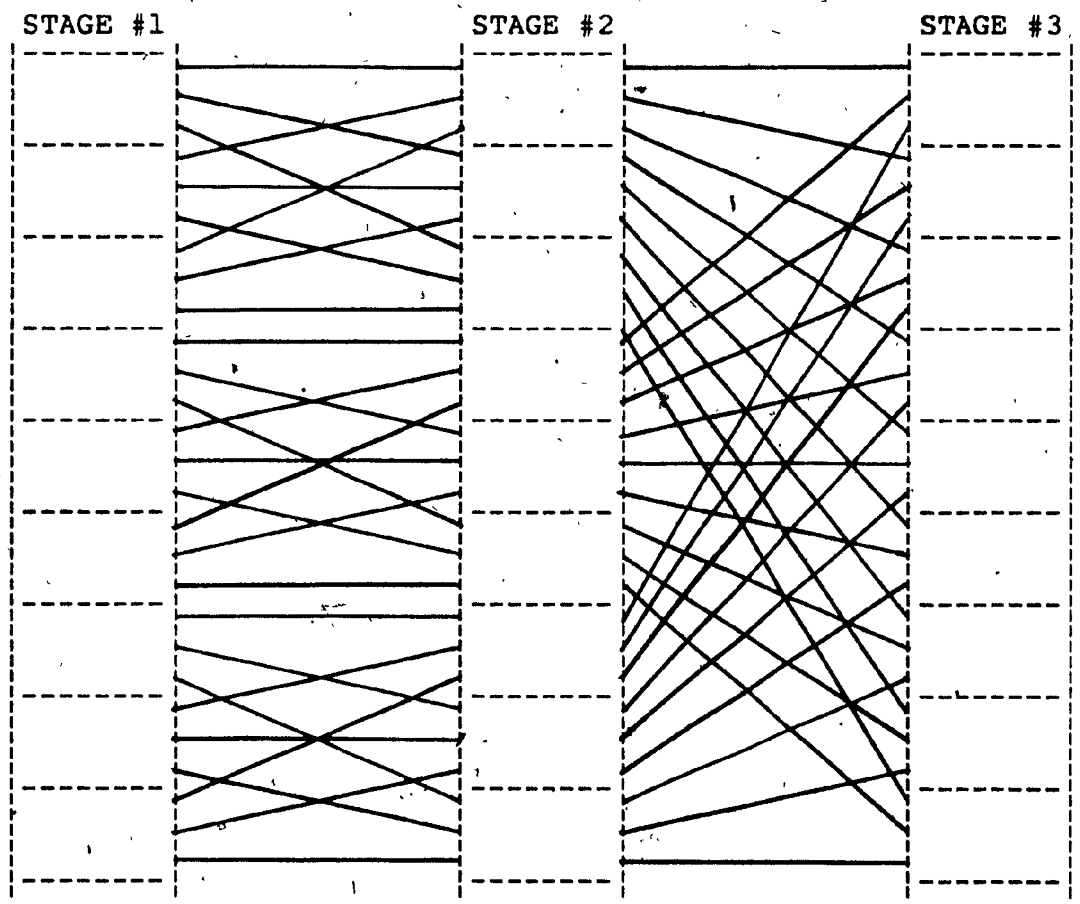


FIGURE 2.3

2.4 Concluding Remarks

Some encryption methods were presented in this chapter for use in enciphering software packages and data. The underlying strategy is to make the encrypting process as irreversible as possible for unauthorized persons. Given time, and using brute force, many intricate cipher texts have been decoded by intruders. Breaking the code may be becoming easier these days with the use of high speed computers. The ease to decipher encrypted code must be met with much more sophisticated encrypting software, as well as tamper-proof hardware.

The methods described in this chapter for decision tables and text encryption are similarly effective for the job they set out to do. Either of these methods can become as sophisticated as the user wishes them to be. Time, money, and inginuity are among the main catalysts needed to produce tamper-proof encryption proceedures. These methods when applied to hide the logic of programs, then one can establish legal propriety of the program in case of a dispute.

CHAPTER 3

MINIMUM COST TELECOMMUNICATION NETWORK DESIGN

3.1 Preamble

Our thesis now departs from the domain of static information, discussed in reference to software packages and data, and moves into the realm of data telecommunication.

We are concerned here with communication networks, or graphs, that are composed of data terminals and the communication links, circuits or media, that enable the users to transmit data from one terminal site to another.

As outlined in the introduction section of this thesis, these terminal sites may exist almost anywhere. A network may be set up within a building, a city, a country, or globally.

Our main concern is, given an existing network or some original design of a proposed network, to find the minimum cost network plan. We did not concern ourselves with the costs involved at the different sites, for, these costs usually do not depend on the topology. We are interested in reconfiguring a network to eliminate all unrequired communidation links, but still enabling every node to communicate with every other node, if such is the mandate of the network. In many instances, there may always be a high volume of data between certain sites. Also, direct links may have to exist connecting sites transmitting highly classified information. These links can consequently be made secure against unauthorized persons. In both these cases, direct links between the terminals in question is a necessity, even if it renders the network cost-inefficient.

The telecommunication costs between terminals, amongst other factors, are directly proportional to the distance between them. This may not always be the case, for with today's tariffs, techniques, competing carrier companies, instances may be found where it costs more to transmit between two terminals that are closer together than between another two that are further apart.

Two different costs between nodes that are used in our analysis are:

(1) Total actual communication costs. This is a fixed cost associated with a link in the network, and does not vary,

(2) Average cost. This cost is computed using the frequency of transmission or character rate, and the cost per message, per character, or per unit of time.

When there are only two nodes in the graph, there is nothing more one can do to render the network any more cost-efficient; the two nodes are simply joined together by a direct single link.

FIGURE 3.1

When a third node is added, inefficiencies are introduced if the three are linked in a certain manner. Figure 3.2 shows an efficient set-up where all parties may com-

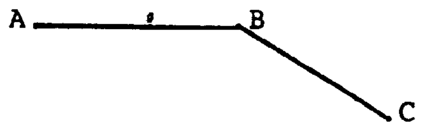


FIGURE 3.2

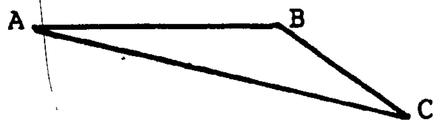


FIGURE 3.3

municate with each other. Users A and C encounter a bottleneck since their transmissions to each other must pass through user B. The Network depicted in Figure 3.3 is not a minimum cost network since a cycle exists. User A may send to user C by two different routes, A,C or A,B,C. In our analysis we set out to determine minimum cost networks by eliminating such cycles.

3.2 Analysis

Two different networks were analyzed for minimum cost. The small network had 10 edges, while the larger had 100. For each of these networks a different cost was used. In the case of the first, the total actual communication cost between two nodes was used. The cost used in the second was an average cost computed using the frequency of transmission and the cost per transmission. Number of characters and a cost per character, as well as time to transmit and transmission cost per unit of time could have been used just as easily.

```
Cost of Link = No. of Messages * Cost/Message
             = No. of Characters * Cost/Char.
             = No. of Chars. * Trans. Time/Char.
                                  * Cost/Unit of Time
```

Input to the algorithm was a record consisting of:

(1) 2 vertices of the graph depicting the nodes of the two sites that were directly linked,
(2) either the total actual fixed cost of transmitting, or
(3) the average cost per message, and
(4) the frequency of transmitting.

To construct a minimum cost tree, the following procedure was used.

After the records were read into an array, HEAP SORT was used to sort the records into a nondecreasing cost of communication sequence. After sorting was completed, edges (v,w) were accessed in the nondecreasing cost of communication sequence and added to a tree T. If either of the two vertices did not already exist in T, a new subtree was started. A search of all subtrees created so far was made to determine if both vertices of the edge being added to T were within the same subtree. If both vertices were found to be in the same subtree, that is, both had the same subtree ROOT, the edge was discarded and not added to the subtree, thus eliminating the creation of a cycle. For example, if a subtree with root 1 contains the edges {1,2,3,4,5}, and the next edge to be analyzed for possible addition to the above subtree is (3,5), this edge would not be added since both vertices 3 and 5 already occur. If, on the other hand both vertices did not occur in any of the so-far-created subtrees, the edge (v,w) was then added to the subtree.

After all vertices were looked at, the subtrees were all 'united' under one tree T. Since the edges were taken in an increasing cost of communication order and those causing cycles discarded, the resulting tree can be looked upon as a minimum cost network.

The original and resulting minimum cost networks analyzed, may be seen in Figures 3.4 to 3.7 inclusive.

Minimum Cost Network Algorithm

```
                      PROGRAM MCNA;
    /*              Initialization                                   */
line  1           N <-- no. of edges;
line  2           FOR I=1 TO N BY 1
                      DO;
line  3               R(I)=I:
line  4               COUNT(I)=1;
line  5               VERTEX(I,1)=1;
                      END;
    /*  Read in both vertices and cost of transmission               */
line  6           READ (V(I),W(I),COST(I), FOR I=1,N BY 1);
    /* Upon return from HEAP SORT, RECSORT contains the              */
    /*   record indices of the n records in a nondecreas- */
    /*     ing order of edge cost                           */

line  7           CALL HSORT(N,RECSEQ);
line  8           FOR I=1 TO N BY 1
                      DO;
    /* Find root of sub tree containing the first vertex   */
line  9               ROOT1 <-- V(RECSEQ(I));
line 10               CALL FIND(ROOT1);

    /* Find root of sub tree containing the 2nd. vertex    */

line 11               ROOT2 <-- W(RECSEQ(I);
line 12               CALL FIND(ROOT2);
    /* If both vertices are in the same set, discard, so   */
    /* that a cycle will not be formed.                    */
line 13              IF ROOT1 ≠ ROOT2
                     THEN DO;
line 14                   WRITE (V(RECSEQ(I)),W(RECSEQ(I)));
line 15                   TOTCOST = TOTCOST + COST(RECSEQ(I));
     /*  Unite existing sub trees, or start a new one      */
line 16                   CALL UNION (ROOT1,ROOT2);
                          END;
                     END;


                PROCEDURE FIND (ROOT);

/* Find the root of the tree containing the element i,      */
/* using the collapsing rule to collapse all noods from i   */
/* to the root j.                                           */

      j <-- i;
      WHILE PARENT(j) > 0              /* Find the root    */
            DO;
            j <-- PARENT(j);
            END;
      k <-- i;
      WHILE k ≠ j     /* Collapse nodes from i to root j*/
           DO;
           t <-- PARENT(k);
           PARENT(k) <-- j;
           k <-- t;
           END;
      RETURN(j);
  END FIND;
       PROCEDURE UNION(ROOT1,ROOT2);

         IF COUNT(ROOT1) < COUNT(ROOT2)
         THEN DO;
              K <-- COUNT(ROOT1);
              FOR J = 1 TO K BY 1
                  DO;
                  VERTEX(ROOT2,(COUNT(ROOT2)+1))
```

```
                                       <-- VERTEX(ROOT1,J);
                   COUNT(ROOT2) <-- COUNT(ROOT2) + 1;
                   COUNT(ROOT1) <-- 0;
                   RETURN;
                   END;
               END;
          ELSE DO;
               K <-- COUNT(ROOT2);
               FOR J = 1 TO K BY 1
                   DO;
                   VERTEX(ROOT1,(COUNT(ROOT1)+1))
                                       <-- VERTEX(ROOT2,J);
                   COUNT(ROOT1) <-- COUNT(ROOT1) + 1;
                   COUNT(ROOT2) <-- 0;
                   RETURN;
                   END;
            END;
                      END UNION;

                 END MCNA;
```

The following is a time analysis of the minimum cost analysis program MCNA.

The lines of program MCNA that entered into our analysis, have been marked on the left with their respective line numbers for easy reference.

TABLE 3.1

PROGRAM MCNA TIME ANALYSIS CHART

LINE NO.	TIME IN TERMS OF n TO EXECUTE
1	1
2	(n + 1)
3 to 5	n
6	1
7	O(nlog n)
8	(n + 1)
9	n
10	O(log n)
11	n
12	O(log n)
13	n
14	(worse case with no cycles at all = n)
15	n
16	O(n)

Consequently, we can say that the time required for program MCNA to execute is O(nlog n).

If n = 10, the cost is:

$$x = C(10\log_2 10),$$

where C is a constant.

Now, if n = 100, the cost becomes:

$$y = C(100\log_2 100),$$

where C is the same constant.

We have, $$\frac{x}{y} = \frac{(10/\log_{10} 2)}{(200/\log_{10} 2)}$$

$$x/y = 1/20$$

That is, $y = 20x$

Table 3.2 shows the results obtained when minimizing networks of 10 and 100 edges. The costs associated with edges in a network are not shown in Table 3.2 below. However, we make use of these costs in obtaining minimum cost networks.

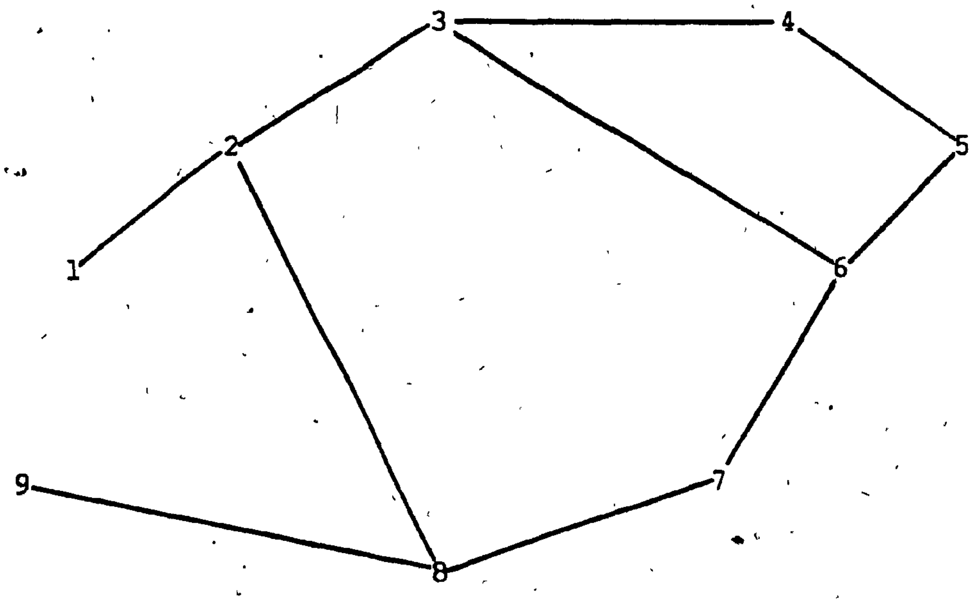
TABLE 3.2

TYPE OF COST	NO. OF EDGES BEFORE	NO. OF EDGES AFTER	PROCESSING TIME (C.U.)	ORIG. COST	NEW COST	COST SAVINGS
ACTUAL	10	8	455	$ 550.	$380.	$ 170.
ACTUAL	100	14	1533	5050.	415.	4635.
AVRG.	10	8	485	3850.	2400.	1450.
AVRG.	100	14	1585	338350.	23247.	315103.

The 'processing time' is given in CPU UNITS, where 1 CPU UNIT = 26.04 micro seconds.

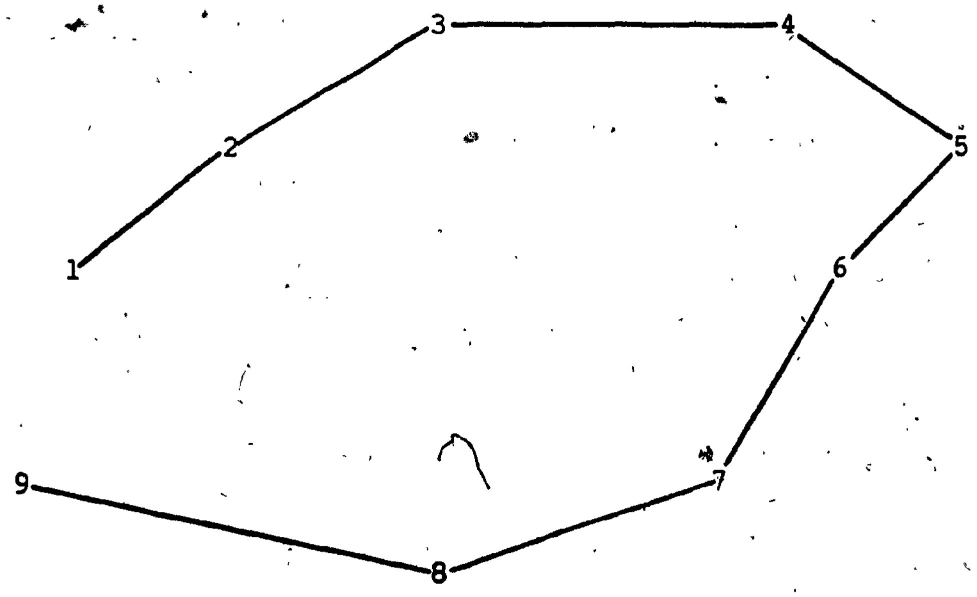
From the above results, it took 1533/455, or 3.4 times as long to process 10 times the number of graph edges, a far cry from the theoretical 20 times. In the case of using the average cost of an 'edge' to compute a minimum cost network, it also took 3.4 times longer to process the 100 edges.

The following 4 figures depict the two networks of 10 and 100 edges before the least cost design was applied and what they looked like afterwards.



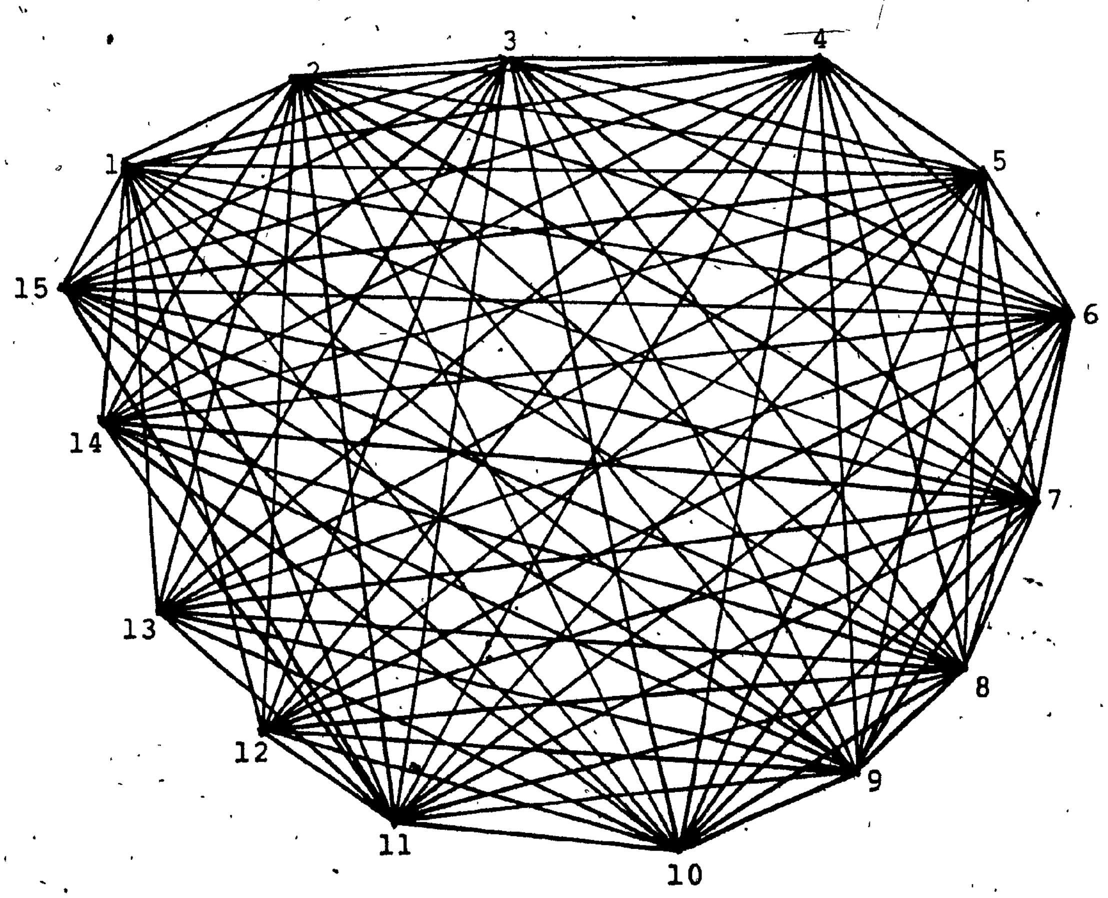
n = 10 Edges. Before minimum cost design.

FIGURE 3.3



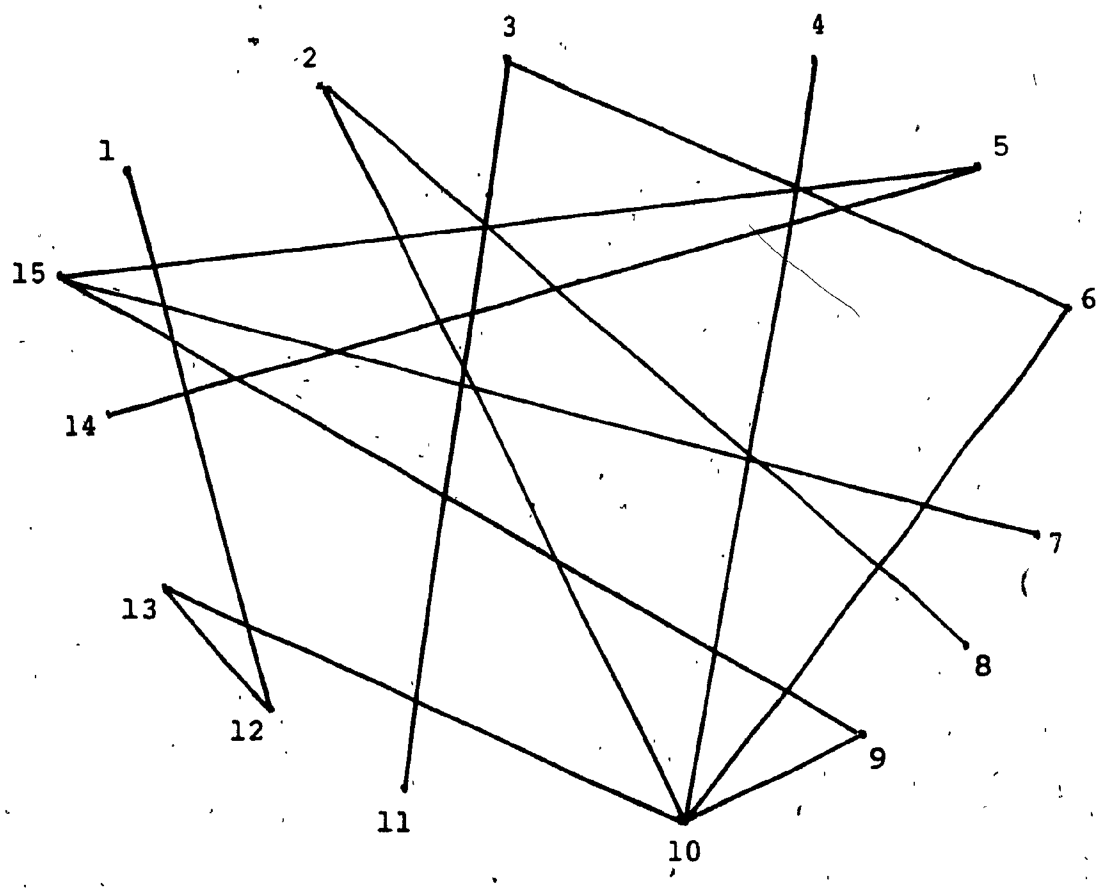
n = 8 Edges. After minimum cost design.

FIGURE 3.4



n = 100 Edges. Before minimum cost design.

FIGURE 3.5



n = 14 Edges. After minimum cost design.

FIGURE 3.6

3.3 The Mix

The mix is described by David L. Chaum {4} as being a message switching computer. An analogy may be drawn up between a railroad roundhouse and a mix. In the former, the table turns until the correct track is lined up for the train to proceed, while in the latter, the computer after scanning and editing the addressee information, selects the proper circuit to send the transmission; a sort of message switcher. Fortunately, the mix's role is much more than what was just mentioned. In Chapter 5 the role of the mix in network security is discussed.

The mix must be mentioned here since it is a part of the communication network. This is not always the case as D.K. Branstad {1} outlines. The mix may be thought of as being exterior to the network, or networks. In this case, the services of the mix would be used when access to a different type of network is desired. Firstly, there are the many carrier companies from which one must choose. The telephone company, Canadian National/Canadian Pacific Telecommunications, RCA, Western Union, etc., all offer data carrier services, but may not all have the same transmission format. There is still need to support the old teletype 5 and 8 level start/stop codes. Then there is binary synchronous transmission, BYSNC, and the more recent X.25, and Synchronous Data Link Control, SDLC, that one must contend with.

What if a transmission must cross many carrier territories of different transmission types to reach its destination? Not only does one have to contend with differences in transmission formats, but also with the different terminal types. Some accept fewer characters per line than others. The speed of the terminals is also different from network to network. In all these cases, the mix can provide homogeneity in a heterogeneous environment.

In Chapter 5 we will talk about a very important function and role that a mix has in the network. We are here speaking of the security function and the role of the authentication server.

Depending on the design, size, and security considerations of the network, there could only be one computer serving all the terminals, or many mixes, wherein a particular mix could serve one or many servers.

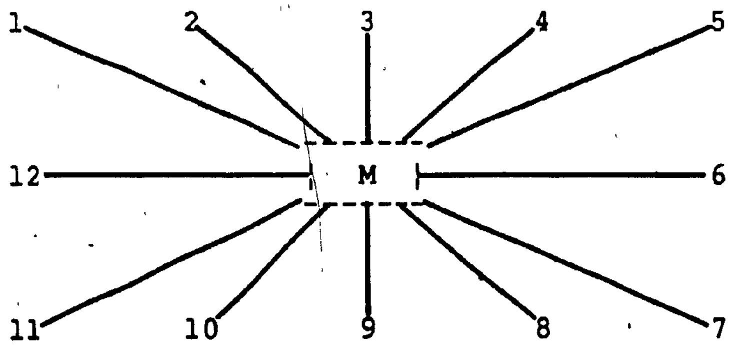


FIGURE 3,7

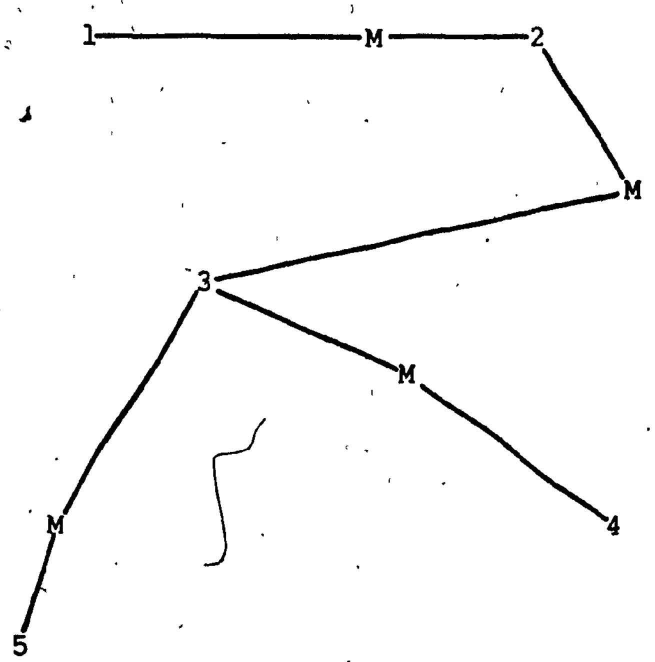


FIGURE 3.8

The cost of the network would of course have to include the cost of these mixes. The 'over kill' would occur when each terminal is associated with its own private mix. This design would be extravagant and very inefficient. We therefore had to come up with some method that would reduce the number of mixes to a minimum, but without jeopardizing the security of any user and the network as a whole.

For the analysis of the minimum number of mixes, we define a graph G with a set of vertices V. D is also a graph whose vertices are a subset of V. We discuss two types of graphs:

(1) Dominating Set

A subset D of V is called a dominating set if every vertex w in (V-D) is adjacent to one or more vertices in D.

(2) Minimal Dominating Set

A dominating set D is said to be minimal, if no subset of D can be a dominating set.

Our goal therefore, was to choose the minimal dominating set of minimum size. Once found, mixes would then be installed at these nodes.

To come up with an algorithm that would give all the dominating sets, is quite a difficult problem. The number of dominating sets increases exponentially as the number of edges of a graph are increased.

We were able to arrived at a heuristic algorithm that produces one dominating set whose size is probably close to a minimum. The algorithm was not tested exhaustively, but only on small sets. The results obtained were encouraging.

The main points of this algorithm follow:

A search is made looking for the vertex adjacent to the greatest number of edges. This vertex is then designated as a node where a mix will be established. This node as well as all vertices adjacent to this node, are marked for no further examination. The above is repeated until all vertices have been looked at.

The output of this algorithm is a dominating set, that is, a list of the nodes that will sustain the mixes.

An example.

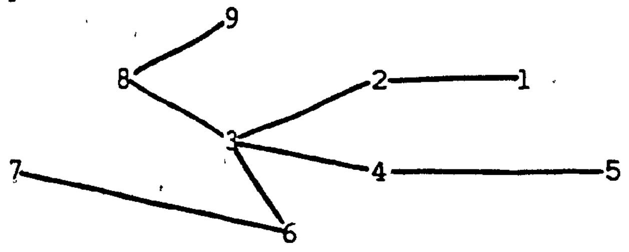


FIGURE 6.9

The vertices of Figure 6.9 selected by the algorithm to sustain mixes were vertex 2, 4, 6, and 8. The following proves that set D {2,4,6,8} is a dominating set.

```
            V  =  { 1, 2, 3, 4, 5, 6, 7, 8, 9 }
            D  =  {    2,    4,    6,    8    }
      ---------------------------------------------
     (V - D) =  { 1,    3,    5,    7    9 }
```

SELECTED VERTEX IN (V-D)	SELECTED VERTEX IN (D)	EDGE FORMED	DOES EDGE EXIST IN (V)
1	2	1,2	YES
3	8	3,8	YES
5	4	5,4	YES
7	6	7,6	YES
9	8	9,8	YES

Since every vertex in (V-D) along with at least one vertex in D form an edge in the original graph V, it may be said that the set (V-D) is a dominating set.

Also, we can verify that D does not have a subset that would also be a dominating set. Consequently, D is a minimal dominating set. If we placed mixes at nodes 2, 4, 6, and 8, then a transmission would have to go through at least one mix to reach another node.

3.4 Concluding Remarks

One of the main objectives of obtaining a network topology is to minimize the cost and risk involved. Only by minimizing the risk we can assume security of communication. Some of the links in the network might be absolutely necessary either for reasons of security or volume. Given a complete characterization of a network in terms of cost parameters, we can obtain a minimum cost network. We have introduced the concept of mixes for proper, efficient and secure functioning of a network.

Combining the two main sections of this chapter, namely, finding a minimum cost network as well as finding the minimim number and position of mixes, one can produce a design of a least cost telecommunication network.

CHAPTER 4

SHARING A SECRET IN A HIERARCHY

4.1 Preamble

The following situation is considered: There exists a company that has five top executives. These executives are the only personnel allowed to access a particular classified file in auxiliary memory, or logon to a highly secured telecommunications network. For each of these, there exists an authentication interface that permits or disallows access. This interface could be software, or the company could have chosen the authenticating logic be hard wired in the terminal itself. Some of the problems with imbedded wired logic are the much higher costs incurred when changes are necessary, as well as the longer time the changes usually take to complete.

The criteria for successful satisfaction of the authentication logic, is that the co-operation of at least two executives is needed. They must enter their correct identities and passwords in order to activate the unlocking mechanism to permit them access. The system expecting the credentials of at least two persons, would thus require that at least two executives enter in agreement for fraudulent use of the data or network. The minimum number of participants can even be increased to four or five. Another arrangement may be that the unlocking logic expects that the codes of one or two particular executives will always be present besides a combination of the rest of the authorized members. The higher the minimum requirements, hopefully the lesser the chance of coercion and corrupt misuse of the facilities.

If a minimum of two is taken, there are a total of:

$5C_5 + 5C_4 + 5C_3 + 5C_2 = 26$ different combinations of the logon codes the authentication logic would have to cope with for validating the input. We discuss several methods of authentication logic below.

Mere comparison of alphanumeric characters by the sign on procedure in order to validate the executives' credentials, would render their codes vulnerable to open attacks either in main storage or just through brute force repeated attempts. Some bit or character manipulation using all the input is necessary. The codes would be manipulated in such a way that they do not resemble the original input made by the executives. The more dissimilarity the raw input has with the final character string just prior to validation, the more difficult it would be for an impostor to trace the codes back to their original value.

4.2 Method A - Similar Bit Structure

Each executive has an alphanumeric code assigned to him. Each code is of the same length and ends with an alphabetic character directly related to the executive's identity. Bit manipulation of the alphabetic characters and the identities renders the two values equal, no matter what the combination of executives is that are taking part. An example follows:

En - refers to the nth executive

ID - refers to an executive's identity

CD - refers to an executive's code, or password

TABLE 4.1

EXECUTIVES' IDENTITIES AND CODES

EXECUTIVE	BIT STRUCTURE OF IDENTITY	NIBBLE	CODE	BIT STRUCTURE OF LAST CHAR OF CODE
E1	1111 0001	0001	MONA	1100 0001
E2	1111 0010	0010	CURB	1100 0010
E3	1111 0011	0011	LARC	1100 0011
E4	1111 0100	0100	BIRD	1100 0100
E5	1111 0101	0101	#%5E	1100 0101

As stated before, no less than two individuals must take part in the unlocking procedure. The terminal entries for executives 1, 3, and 5 would be as follows:

EXAMPLE 4.1

```
3
ID=5
CD=#%5E
ID=1
CD=MONA
ID=3
CD=LARC
```

All these entries would either be heavily over printed or not typed by the terminal. In the case of video displays, the typed in characters would be entered into a non-display field. In this way other persons in view of the terminal's output would not get to see the other executives' identities and passwords. The number 3 at the beginning of the sign-on procedure signifies to the system how many participants are taking part in the session. This entry could very well have been eliminated by having the software do the counting, but it was left there to add to the complexity for signing on.

The bit manipulation is as follows:

STEP 1 - The bit manipulating logic takes the 4 bits from the second nibble, of each ID, that is, the second half of the byte, reading from left to right, and concatenates them. In this example, the result would be:

```
0101 0001 0011
 5    1    3
```

STEP 2 - The first 3 alphanumeric characters are stripped off from each code. The codes would then become E, A, and C.

STEP 3 - The first nibble from each of the characters of STEP 2 would be eliminated. This would give:

```
0101 0001 0011
 5    1    3
```

STEP 4 - The result of STEP 1 is compared with the result of STEP 3. If equal, security has been satisfied and access is permitted. Consequently, for a successful access the following criteria must be met:

(1) No less than two executives taking part,
(2) Identities must be correct,
(3) Codes must be correct.

The memory requirements for method A are as follows. Since the comparison for the validation step is done on two different portions of the input, namely the identity and the password code, no constants need be retained for this step. Therefore number of memory locations is NIL.

4.3 Method B - Determinants

Definitions:

A SET refers to a matrix made up of the same number of executives but different combinations thereof,

COMBINATIONS refer to matrices made up of different numbers of executives.

EXAMPLE 4.2.

For example $\begin{bmatrix} E1 & E2 \\ a & b \end{bmatrix}$, $\begin{bmatrix} E3 & E5 \\ c & d \end{bmatrix}$

are matrices of the same SET since they both contain two executives each, while,

$$\begin{bmatrix} E1 & E5 \\ g & h \end{bmatrix}, \quad \begin{bmatrix} E2 & E4 & E5 \\ 2E2 & 2E4 & 2E5 \\ 4E2 & 4E4 & 4E5 \end{bmatrix}$$

are COMBINATIONS since they contain different numbers of executives. The first matrix contains two executives, while the second contains three executives.

Matrices are constructed according to the number of executives taking part in the unlocking session. The extra rows needed to complete a matrix are computed by taking multiples of the rows just above them. In our analysis, these are referred to as the completing elements. The determinant of the matrix is evaluated and the result is compared to the correct result stored in the computer. If equal, unlocking takes place, while if unequal, access is rejected.

Our main concern using these matrices, was the overall security of the system. We wanted to maintain as few constants in the system as was possible. The fewer the constants to be maintained in the system, the easier they are to hide from an intruder. Also, maintenance of the system is easier and faster.

The first option to be analyzed was where the value of the determinants remain the same, while the value of the x's, that is, the completing elements differ from SET to SET and COMBINATION to COMBINATION. The value of the x's is the same within a single matrix.

In Example 4.3 below, executives with their identity numbers E1 and E4 attempt access to classified information. While, in Example 4.4 executives with identity numbers E3, E4, and E5 endeavour to gain access to the classified files or network. In the examples below, x_1 and x_2 are system parameters.

$$D_1 = \begin{vmatrix} E1 & E4 & x_1 \\ 2E1 & 2E4 & x_1 \\ x_1 & x_1 & x_1 \end{vmatrix} = Y$$

EXAMPLE 4.3

$$D_2 = \begin{vmatrix} E3 & E4 & E5 & x_2 \\ 2E3 & 2E4 & 2E5 & x_2 \\ 4E3 & 4E4 & 4E5 & x_2 \\ x_2 & x_2 & x_2 & x_2 \end{vmatrix} = Y$$

EXAMPLE 4.4

We insist that:

$D_1 = D_2$ and x_1's $\neq$ x_2's, but the completing elements within each respective matrix are the same.

The computer storage requirements for this first option are twenty-seven memory positions. One position for the result Y, and twenty-six for the different x's.

The second option analyzed was where the value of each determinant differs, but the value of the completing elements, that is, the x's, remain the same for all SETS and COMBINATIONS.

$$D_3 = \begin{vmatrix} E3 & E4 & x \\ 2E3 & 2E4 & x \\ x & x & x \end{vmatrix} = A$$

EXAMPLE 4.5

$$D_4 = \begin{vmatrix} E3 & E4 & E5 & x \\ 2E3 & 2E4 & 2E5 & x \\ 4E3 & 4E4 & 4E5 & x \\ x & x & x & x \end{vmatrix} = B$$

EXAMPLE 4.6

We insist that:

D_3 is $\neq D_4$, but the completing elements of both determinants are all the same.

The memory requirements for the second option are as follows:

One position is required for the completing elements since they are the same throughout each SET and

COMBINATION, while twenty-six locations, one for each determinant result, are required for a total of twenty-seven.

For the third option, we insisted that the values of all the SETS and COMBINATIONS of determinants, differ. The completing elements are the same only within each SET, but differ from other SETS of different combinations.

$$D_5 = \begin{vmatrix} E1 & E2 & x_2 \\ 2E1 & 2E2 & x_2 \\ x_2 & x_2 & x_2 \end{vmatrix} = E$$

EXAMPLE 4.7

$$D_6 = \begin{vmatrix} E4 & E5 & x_2 \\ 2E4 & 2E5 & x_2 \\ x_2 & x_2 & x_2 \end{vmatrix} = F$$

EXAMPLE 4.8

$$D_7 = \begin{vmatrix} E2 & E4 & E5 & x_3 \\ 2E2 & 2E4 & 2E5 & x_3 \\ 4E2 & 4E4 & 4E5 & x_3 \\ x_3 & x_3 & x_3 & x_3 \end{vmatrix} = G$$

EXAMPLE 4.9

$$D_8 = \begin{vmatrix} E1 & E3 & E5 & x_3 \\ 2E1 & 2E3 & 2E5 & x_3 \\ 4E1 & 4E3 & 4E5 & x_3 \\ x_3 & x_3 & x_3 & x_3 \end{vmatrix} = H$$

EXAMPLE 4.10

We insist that the value of the determinants are not equal. That is:

$$D_5 \neq D_6 \neq D_7 \neq D_8 .$$

The completing elements of matrices of the same SETS are equal. That is to say:

x_2's of D_5 = x_2's of D_6 since same set.

x_3's of D_7 = x_3's of D_8, again since same set.

It must be noted that it would be beneficial that the completing elements from the different SETS should not be the same in order that a high degree of complexity be maintained in this system.

This option requires the following number of memory areas:

The number of locations required for the different values of the determinants is twenty-six. We need thirty-two more locations to maintain the completing elements. Therefore, a total of 26 + 32 = 58 memory locations are needed for this option.

The fourth option analyzed is as follows:

The value of the various determinants differs. The value of the completing elements differ for the different SETS as well as for the different COMBINATIONS.

The memory requirements for this option are, twenty-six locations for the results and one hundred and seventy-six for the completing elements, for a total of two hundred and two memory positions.

A summary of the storage requirements for all the preceding four options is given below in Table 4.2.

TABLE 4.2

SUMMARY OF MEMORY REQUIREMENTS

	OPTION #1	OPTION #2	OPTION #3	OPTION #4
Determinent Value	1	26	26	26
Completing Elements	26	1	32	176
TOTALS:	27	27	58	202

The main goals here are to make it as difficult as possible for an intruder to crack the security system. Also, the authentication system is to use as few memory locations as possible for the completing elements and values of the determinants. The program that accepts the input, sets up the determinants, and calculates their result, must itself be encrypted for total security. Since the same program would be used for all options, the number of memory locations it requires was not entered into the comparisons.

The first and second options require the least number of memory locations. Eventhough either one of these two could be chosen, Options 3 or 4 could appeal more to someone who may think they may be slightly more 'confusing' to the intruder.

4.4 Method C - Coprimes and Multiplicative Inverse

As in the previous methods, we attempt in this method to create complexity for 'confusion', with as little usage of memory for storing constants, as possible.

This method also requires the executives to enter a 4-character password code. The EBCDIC 8-bit configuration code for the following characters is:

TABLE 4.3

EBCDIC 8-BIT CODE

A:	1100 0001	S:	1110 0010
F:	1100 0110	T:	1110 0011
M:	1101 0100	X:	1110 0111
P:	1101 0111	Y:	1110 1000
R:	1101 1001		

The passwords for executives E2, E3, and E5 are XYTR, PSRM, and ATSF respectively.

The top half of each byte of each character is stripped off and the result is concatinated with the rest of the lower nibbles of the executive's code. The 16-bit strings so formed from each executive's code are then binary added together.

```
E2:  XYTR  0111 1000 0010 1001
E3:  PSRM  0111 0010 1001 0100
E5:  ATSF  0001 0011 0010 0110
------------------------------
  RESULT = 1111 1101 1110 0011
```

EXAMPLE 4.11

In the following analysis, the total of the above binary addition will be referred to as RESULT.

In the next step, we calculate the RESULT modulo the first prime number smaller than the number formed with the code AAAA. That is, we find:

$$(\text{RESULT}) \bmod M_1 = a.$$

For a further encryption, the multiplicative inverse of 'a' is computed:

$$a * b = 1 \bmod M_2$$

where 'b' is the multiplicative inverse of 'a', and M_2 is another modulus. 'a', 'b', and M_2 are co-primes. That is, the $\gcd(a,b,M_2) = 1$.

Once the values RESULT, 'a' and finally 'b' are computed, the value of 'b' is used to activate the unlocking mechanism to either permit or refuse access. Twenty-six values of 'b' that permit access are stored if there are five executives and a minimum of two are required to take part in in any unlocking session.

The advantages of this system are as follows: Each combination of executives is associated with a number 'b'. M_1 and M_2 are kept secret and are also stored. There is no need to keep different modulo numbers for each combination of executives taking part. M_1 and M_2 remain the same for all the types of entries. With twenty-eight memory locations needed to store M_1, M_2 and all the values of 'b', there is minimal data to worry about being uncovered by unauthorized persons hoping to gain access into the system. The required parameters would be entrusted only to key personnel, who would make them available to the system just prior to an access attempt.

The logic for this method would probably have to be encrypted itself as a further protection.

4.5 Method D - Polynomial Interpolation

This method is outlined by Adi Shamir {18}. The secret to be encrypted, is some data D, like a safe combination, signatures required on company cheques, or secret codes required for accessing communication networks.

The data D is divided into n parts, D_1, D_2,... ....D_n in such a way that: 1) knowledge of any k or more D_i pieces makes D very easily computable,

2) knowledge of any (k - 1) or fewer pieces, leaves D completely undetermined. We assume that the data D can be converted into, or associated with, a number. A random (k - 1) degree polynomial was chosen:

$$q(x) = a_0 + a_1 x + a_2 x^2 + \ldots\ldots + a_{k-1} x^{k-1}$$

Now, we let:

$$a_0 = D,$$

and then evaluate:

$$D_1 = q(1),\ldots.,D_i = q(i),\ldots.D_n = q(n)$$

Given a subset of k of these D_i values, together with their identifying indices, we can find the coefficients of q(x) by interpolation, and can then evaluate $D = q(0)$.

Knowledge of just (k - 1) of these values, does not suffice to evaluate D. An example follows.

EXAMPLE 4.12.

The number of participants is k = 3. Analysis for fewer or more participants may be found in later sections of this chapter.

Executives: x_1, x_2, x_3, x_4, x_5

Indices, or ID's: $x = 1 \quad 2 \quad 3 \quad 4 \quad 5$

Polynomial of degree $(k - 1)$: $y = ax^2 + bx + c$

Original Polynomial: $y = 2x^2 + 3x - 3$

$y(1) = 2 + 3 - 3 = 2$

$y(2) = 8 + 6 - 3 = 11$

$y(3) = 18 + 9 - 3 = 24$

$y(4) = 32 + 12 - 3 = 41$

$y(5) = 50 + 15 - 3 = 62$

The keys 2, 11, 24, 41, and 62 are given to the respective persons. Not needed any longer, the original polynomial $y = 2x^2 + 3x - 3$ is discarded. Each time an unlocking is to take place, three (key, ID) pairs are required.

Assume x_1, x_2, and x_3 wish to participate in an unlocking session.

Since 3 persons will be taking part,

$$y = ax^2 + bx + c$$

Required - to find a, b, and c.

$x = 1 \qquad y(1) = 2 = a + b + c$

$x = 2 \qquad y(2) = 11 = 4a + 2b + c$

$x = 3 \qquad y(3) = 24 = 9a + 3b + c$

Solving these three simultaneous equations, we get $a = 2$, $b = 3$, $c = -3$.

Substituting these coefficients in

$$y = ax^2 + bx + c$$

we arrive at the original polynomial,

$$y = 2x^2 + 3x - 3$$

To unlock, $y(0)$ must be calculated,

$$y(0) = -3$$

In this same fashion, any other three participants taking part would produce the same result.

4.5.1 Option A - Increase in Number of Participants-1

If for some reason(s) it is felt that the minimum number required to activate the unlocking mechanism is too low, the following is necessary to increase the minimum, and, at the same time continue to require the co-operation of those entrusted with the ability to access the classified files or communication network. The difference now, is that more executives must take part.

For our analysis we chose to increase the minimum from $k = 3$ to $k = 4$. An entirely new polynomial is required of degree $(k - 1) = (4 - 1) = 3$.

$$\text{Let } y = ax^3 + bx^2 + cx + d$$

The five executives' codes remain the same for this analysis, but in practice these may be changed. We illustrate with an example.

Let the original polynomial chosen be

$y = x^3 - 2x^2 + 4x - 2.$

We have
$$\begin{aligned} x &= 1 & y(1) &= 1 - 2 + 4 - 2 = 1,\\ x &= 2 & y(2) &= 8 - 8 + 8 - 2 = 6,\\ x &= 3 & y(3) &= 27 - 18 + 12 - 2 = 19,\\ x &= 4 & y(4) &= 64 - 32 + 16 - 2 = 46,\\ x &= 5 & y(5) &= 125 - 50 + 20 - 2 = 93,\end{aligned}$$

as the codes for executives.

When the first four executives participate, we have to find y(0):

$$\begin{aligned} & & y &= ax^3 + bx^2 + cx + d\\ x &= 1 & 1 &= a + b + c + d\\ x &= 2 & 6 &= 8a + 4b + 2c + d\\ x &= 3 & 19 &= 27a + 9b + 3c + d\\ x &= 4 & 46 &= 64a + 16b + 4c + d\end{aligned}$$

Solving the above 4 simultaneous equations, we obtain, $a = 1,\ b = -2,\ c = 4,\ d = -2$

These coefficients give the resulting polynomial: $y = x^3 - 2x^2 + 4x - 2$

which is the same as the original polynomial. As before, to determine the unlocking code, we compute $y(0)$,

$$y(0) = -2$$

In summary, with an increase in the minimum number, each person's key changes as may the unlocking code. If the value of the constant 'd' of the new polynomial of degree 3 is chosen to have the same value as the constant 'c' of the old polynomial of degree 2, then no change in the unlocking mechanism need be made since the unlocking codes would be the same.

An example to clarify follows.

Minimum number of executives is $k = 3$.

The polynomial chosen is: $y = 4x^2 + 2x - 10$

$y(0) = -10$.

Now the minimum is increased to $k = 4$.

The polynomial chosen is: $y = 9x^3 + 16x^2 - 5x - 10$

$y(0) = -10$, same as above.

Increasing the minimum number of entrusted personnel is not very difficult. A new polynomial of degree one less than the number of participants is required, just as with the old minimum. Once the authentication mechanism, whether it be hardware, or software, is made aware of the new circumstances, namely that a polynomial of higher, or lower degree must be used, the system is ready again for use.

The advantages in increasing the minimum personnel required, gives that extra security to the system. The mere change would make those entrusted with the network think wice before attempting to solicit the (ID,KEY) combination from the other members of the group for reasons to access the facilities for fraudulent use. Besides, a periodically scheduled change in all keys may be advantageous since old keys could have become known to other participating as well as non-participating personnel. For additional security, upon the leaving of an entrusted executive from the firm, it would be very advantageous for the leader of the entrusted group to at least change the codes of the remaining persons in the group , and possibly increase the minimum.

Unfortunately, it would prove a bit costly if the authenticating mechanism was hard-wired, as it would be benefitial but not necessary, that it be changed. The participating personnel's memory may not be good enough to start learning and retaining new codes. Out of habit, old codes may be used causing minor delays that can prove to be crucial in times when quick accessing is required.

4.5.2 Option B - Increase in Number of Participants-2

The following steps describe what is required when the number of entrusted persons is increased:

(1) Call any k of the n entrusted individuals.

(2) Each of these k persons inputs his (ID,Key) pair.

(3) The (k-1) degree polynomial $A_k(x)$ is constructed.

(4) Calculate $A_k(0)$, the unlocking code.

(5) Pick an ID from amongst the x_1 to x_k, and call it x_j.

(6) Excluding the chosen x_j, calculate the following polynomial: $B_k(x) = x(x-1)(x-2)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_k)$.

(7) Compute $A_k(n+1)$ and $B_k(n+1)$.

(8) Choose Y_{k+1} such that $\dfrac{Y_{k+1} - A_k(n+1)}{B_k(n+1)}$ is an integer.

(9) Compute the required polynomial:

$$A_{k+1}(x) = A_k(x) + \frac{B_k(x)}{B_{k+1}(n+1)}\left[Y_{k+1} - A_k(n+1) \right]$$

EXAMPLE 4.13.

As in the Example 4.12, let the original polynomial be: $y = 2x^2 + 3x - 3$,

the executives' ID's be:	1	2	3	4	5,
resulting in the keys:	2	11	24	41	62.

$A_k(x)$ = interpolating polynomial for k pairs

$$= \sum_{i=1}^{k} Y_i C_i \quad \text{where } C_i = \prod_{\substack{i=1 \\ j \neq i}}^{k} \frac{(x - x_j)}{(x_i - x_j)}$$

Number of participants, $k = 3$.

$$A_3(x) = Y_1C_1 + Y_2C_2 + Y_3C_3$$

$$= \frac{Y_1(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} + \frac{Y_2(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} + \frac{Y_3(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

$$= \frac{2(x-2)(x-3)}{(1-2)(1-3)} + \frac{11(x-1)(x-3)}{(2-1)(2-3)} + \frac{24(x-1)(x-2)}{(3-1)(3-2)}$$

$$= \frac{2(x^2-5x+6)}{2} + \frac{11(x^2-4x+3)}{-1} + \frac{24(x^2-3x+2)}{2}$$

$$= x^2 - 5x + 6 - 11x^2 + 44x - 33 + 12x^2 - 36x + 24$$

$$A_3(x) = 2x^2 + 3x - 3$$

The polynomial obtained is the same as the original. This result is obtained no matter which 3 of the 5 executives is chosen.

$A_3(0) = -3$, the same unlocking code as the one in the Example 4.12.

If the number of participants is now changed from $k = 3$ to $k = 4$, the following changes must take place.

Let $x_j = x_2$ and eliminate from the sequence

$$x_1, \ldots\ldots x_i, \ldots x_k$$

Also,

$$B_k(x) = x(x-1)(x-2)\dots(x-x_{j-1})(x-x_{j+1})\dots(x-x_k) \qquad (1)$$

The value (n+1) below means a value out of the range from 1 to n persons.

Now,

$$A_{k+1}(x) = A_k(x) + \frac{B_k(x)}{B_k(n+1)}(Y_{k+1} - A_k(n+1)) \qquad (2)$$

If x_1, x_2, and x_3 are invited to participate, then,

$$A_k(n+1) = C_1Y_1 + C_2Y_2 + C_3Y_3$$

Let (n + 1) = 6.

$$A_3(6) = \frac{Y_1(6-x_2)(6-x_3)}{(x_1-x_2)(x_1-x_3)} + \frac{Y_2(6-x_1)(6-x_3)}{(x_2-x_1)(x_2-x_3)} + \frac{Y_3(6-x_1)(6-x_2)}{(x_3-x_1)(x_3-x_2)}$$

$$= \frac{(2)(4)(3)}{2} + \frac{(11)(5)(3)}{-1} + \frac{(24)(5)(4)}{2}$$

$$A_3(6) = 12 - 165 + 240 = 87$$

Now, from equation (2) we must endeavour to make the following an integer,

$$\frac{Y_{k+1} - A_k(n+1)}{B_k(n+1)}$$

From equation (1), and leaving x_2 out,

we have, $B_3(6) = 6(6-1)(6-3) = 90.$

Therefore, for $\frac{Y_{k+1} - 87}{90}$ to be an integer,

Y_{k+1} must equal 177, since $(177 - 87)/90 = 1.$

From equation (2),

$$A_{k+1}(x) = 2x^2+3x-3 + x(x-1)(x-3)\left[\frac{(177) - (87)}{90}\right]$$

$$= 2x^2 + 3x - 3 + x^3 - 4x^2 + 3x$$

That is, $A_{k+1}(x) = x^3 - 2x^2 + 6x - 3$

$A_4(0) = -3$, the same unlocking code as $A_3(0)$.

We have,

$A_4(1) = 2$, $A_4(2) = 9$, $A_4(3) = 24$, $A_4(4) = 53$, $A_4(5) = 102$

From the previous scheme when the number taking part $k=3$, we had,

$A_3(1) = 2$, $A_3(2) = 11$, $A_3(3) = 24$, $A_3(4) = 41$, $A_3(5) = 62$.

Concluding remarks. With an increase in the number of participants, this method retains the same unlocking code. Those k persons that took part initially, retain their same keys, while those that did not take part, that is the (n-k), must obtain new keys using the new polynomial. Finally, the individual that was left out, must also be entrusted with a new key, again obtaining it by using the new polynomial.

In this example, x_1 and x_3 retain their old codes, namely 2 and 24 respectively. x_2 requires a new one since he was the one that was left out. x_4 and x_5 require new codes since they were among the (n-k) that were not invited to participate.

Some advantages of this system are as follows.

It could very well be that the code of certain entrusted individuals must be changed, due possibly to the leader becoming suspicious of them, but, not suspicious enough to warrant their exclusion from the pact.

If the unlocking mechanism is hard-wired, the fact that the unlocking code does not change, this method will accommodate the increase in the required minimum of entrusted executives, while not requiring the costly hard-wired logic to be altered.

This method decreases the inconvenience for certain personnel by not having to change the code of those with a weak memory.

Since there are now (k + 1) persons that must take part in an unlocking, the possibility of group coercion is somewhat eliminated, but not completely.

On the other hand, slight disadvantages are inherent in this method.

It may cause some inconvenience to those whose code was changed.

Those people whose code was changed may become somewhat perturbed after finding out certain others' code was not changed.

4.5.3 Option C - Increase in Number of Participants-3

A description follows of what is required if use is to be made of the procedure of this section to increase the number of entrusted participants that must take part in an unlocking session.

(1) A selection of k from the n entrusted personnel would be made.

(2) Using their respective codes, or indices, and their keys, the polynomial $A_k(x)$ of degree (k-1) would then be calculated.

(3) A random number G would then be generated, outside the range 1 to n. This would be the new ID, or index.

(4) The key associated with this new index is then calculated. Let it be V. The pair (G,V) is considered as the

(k+1)st. pair of (ID,Key).

(5) Once it has been established that V is not equal to $A_k(G)$, construct $A_{k+1}(x)$.

EXAMPLE 4.14

As in Examples 4.12, and 4.13, let the original polynomial be: $A_3(x) = y = 2x^2 + 3x - 3$,

and $y(1) = 2$, $y(2) = 11$, $y(3) = 24$, $y(4) = 41$, $y(5) = 62$.

Let k, the initial number taking part be three. If k is to be increased to four, a random number is chosen representing a code not previously existing, that is, one not among the x_1 to x_n indices. This will represent the $x_{(k+1)}$ st. person.

The pair (x_{n+1}, Y_{n+1}) must be chosen so that,

$$Y_{n+1} \neq A_k(x_{n+1})$$

Assume n+1 = 6, that is, the 6th. person, and the random key chosen for x_6 was 147. This number was chosen so that the coefficients of the new polynomial $A_4(x)$ would be integers. But, in effect, Y_{n+1} may be any number, except $A_k(n+1)$. Also, (n+1) could also have been any integer, but 1 to n inclusive.

We write,

$$A_{k+1}(x) = A_k(x) + \frac{B_k(x)}{B_k(x_{n+1})}(Y_{n+1} - A_k(x_{n+1})).$$

Chosing persons $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$

$$A_4(x) = 2x^2 + 3x - 3 + \frac{(x-1)(x-2)(x-3)}{(x_{n+1}-x_1)(x_{n+1}-x_2)(x_{n+1}-x_3)} *$$

$$(Y_{n+1} - (2x_{n+1}^2 + 3x_{n+1} - 3))$$

Substituting $x_{n+1} = 6$,

$$A_4(x) = 2x^2 + 3x - 3 + \frac{x^3 - 6x^2 + 11x - 6}{(6-1)(6-2)(6-3)} \ldots\ldots$$

$$* (147 - (72 + 18 - 3)$$

$$A_4(x) = 2x^2 + 3x - 3 + x^3 - 6x^2 + 11x - 6$$

$$A_4(x) = x^3 - 4x^2 + 14x - 9$$

From the original polynomial:

$A_3(0)=-3$, $A_3(1)=2$, $A_3(2)=11$, $A_3(3)=24$, $A_3(4)=41$, $A_3(5)=62$

and, from the new polynomial:

$A_4(0)=-9$, $A_4(1)=2$, $A_4(2)=11$, $A_4(3)=24$, $A_4(4)=47$, $A_4(5)=86$

Using this method to increase the number of entrusted participants, the unlocking code is different. The keys of those that took part, x_1, x_2, and x_3 in our example, remain the same. Using the new polynomial, new codes must be assigned to the (n-k) non-participants.

An advantage of using this method is that not all entrusted personnel's codes change. This could be an opportunity to change those that might not be trusted as much as others. For there to be an increase in the number taking part, it was probably felt that a breach of security was imminent. An additional advantage is that the unlocking function also changes.

The following are some disadvantages when using this system.

If the unlocking mechanism is hard-wired, the expense of having to change it would have to be incurred. The personnel whose key must be changed might find it a hassle to remember a new key. This option could also be dangerous, for the possibility exists that the personnel on the verge of attempting to form a group to access the file or network for unauthorized purposes, could be among the personnel whose key is not ultimately changed.

CHAPTER 5

AUTHENTICATION FOR SECURE COMMUNICATION IN A NETWORK

5.1 Preamble

There are different stages that one must go through prior to completing a transmission to the intended receiver(s). The first stage we have encountered in Chapter 4. Basically, stage 1 concerns the steps involved in logging on to an insecure network. Once the network system has accepted the intended user, stage 2 is encountered. This stage is labelled a 'handshaking procedure', wherein, the system attempts not only to link the parties who wish to communicate with each other, but also endeavours to authenticate the receiver to the sender and vice versa. In this way the sender is assured that his message(s) is being received by the person he originally intended it, while the recipient is also assured that the message(s) he is receiving is being sent by the person he expects, and not some impostor. The guaranteeing of an (originator,message) set is further investigated in the next chapter when we look at digitalized signatures.

The 'handshaking' step is better known as Communication Network Protocol. Two types of protocol were studied, namely, the Conventional, and the Public Key. In the network studied, the concept of authentication servers, or mixes, as discussed in Chapter 3, is included.

The relationship between these authentication servers and terminals in a network is varied. D. K. Branstad [1] describes some different designs of terminal/authentication server networks.

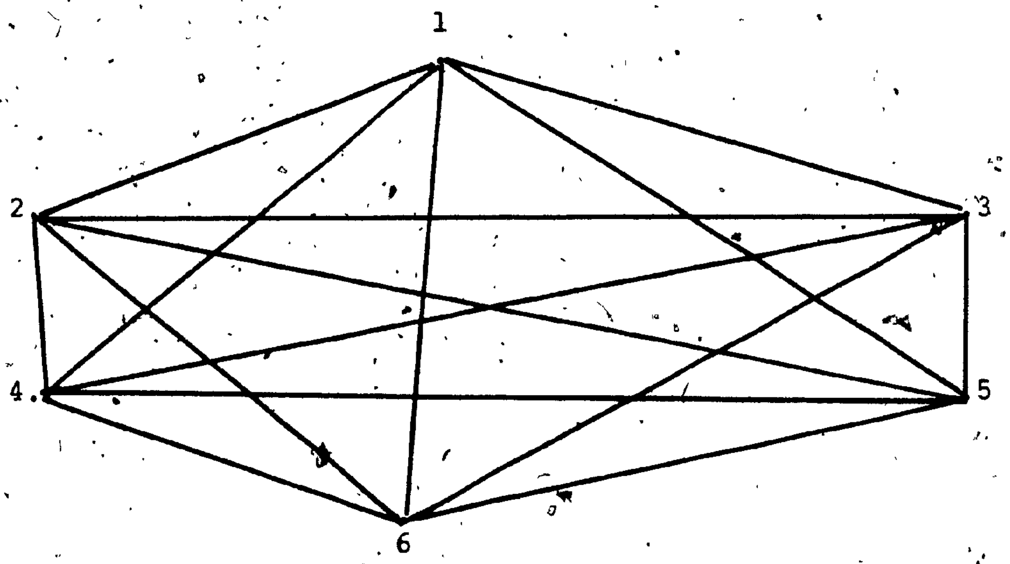


FIGURE 5.1

Figure 5.1 connects every component to every other component. An authentication server may exist one for every terminal, or one shared by two terminals. The network in FIGURE 5.1 due probably to volume and/or priority of messages, may just have to exist as depicted. It is very cost inefficient since all terminals can be reached by more than one path.

In Figure 5.2, every component is connected to one authentication server. This layout only requires one authentication server, since no matter who sends a transmission, it must pass through the centrally located mix.

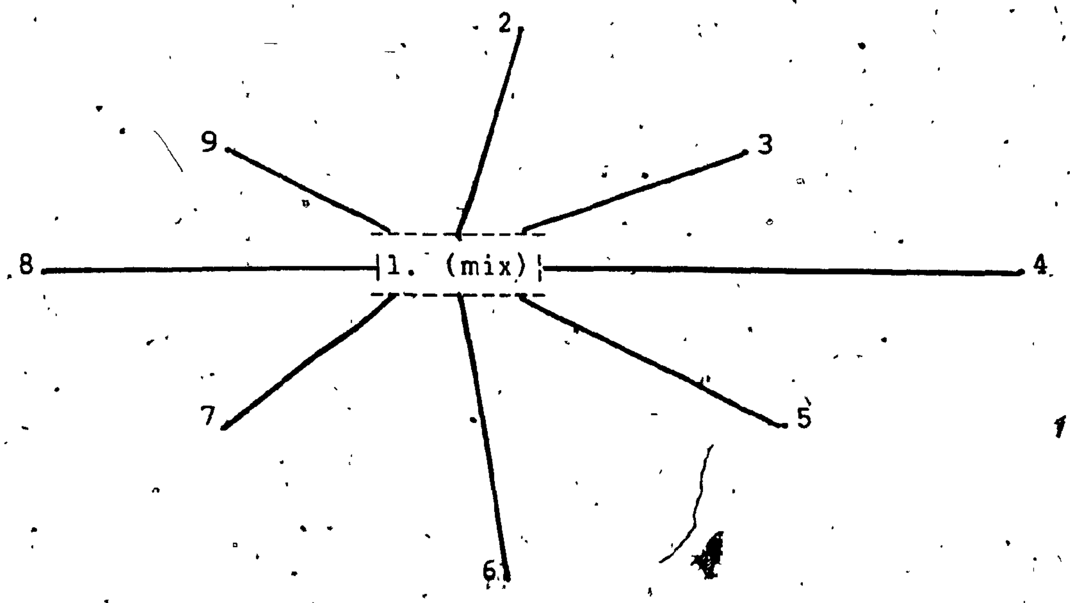


FIGURE 5.2

Depending on the volume, and how secure the server is, this set-up could prove to have a bottleneck at the central point, reducing message throughput. On the other hand, due to the great disparities that exist in the different computers manufactured by different suppliers, and between the different types of computer terminals, such clusters as shown in Figure 5.2 could be looked upon as different networks in themselves, linked only via the interconnection of the various mixes. See Figure 5.3.

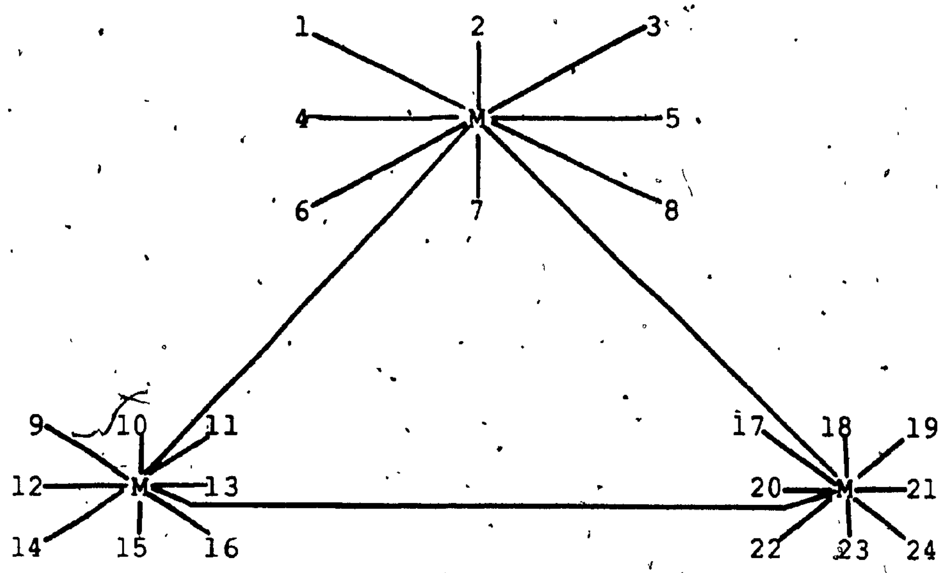


FIGURE 5.3

In the analysis to follow, we wish to bring forth the concept of protocols for secure communication. We felt that the efficient placement of the mixes throughout the network did not enter into this analysis. We merely wanted to emphasize the activity that goes on between:

(1) A sender and his authentication server.

(2) The authentication server of one client and that of another.

(3) The receiver of a message and his own authentication server.

As stated in Chapter 3, along with configuring the terminals and circuits for a least cost network, the placement of the mixes for least cost and maximum security, is also very crucial to a well developed and balanced communications network.

In this analysis we looked at single and multiple authentication servers. Having multiple servers, wherein there is one server per terminal, is not as far fetched as it may sound, for, the authentication operation could take place in some cheap black box, hard-wired especially for this purpose. This apparatus could be part of the terminal itself, or something one can 'snap' on to the circuit.

The parties involved in a communication do not wish that the message content be understood by anyone but themselves while communication is in progress. It may be difficult preventing outsiders from 'listening in', that is, wiretapping, but understanding what has been intercepted is another matter. Consequently, stringent steps would be required to ensure that:

(1) The recipient is totally confident in knowing who the originator is of each message he receives.

(2) The encrypted message along with the 'tools' to encrypt said message, are not understood by a third party.

The 'tools' mentioned above, refer to the following:

(1) An encryption key to encypher the text of the message. This key should be different for each message.

(2) The sender's and recipient's secret keys are known only to these respective persons.

(3) To ensure that no intruder is taking part in the initial protocol, a unique code is generated and appended to the different items of protocol. Examples of such codes are, sequence numbers, a time stamp, etc.

5.2 Conventional Protocol

5.2.1 Single Server

Basically, the conventional encryption method requires the sender of a message to:

(1) Retrieve a unique text encryption key from a central source.

(2) Send the above encryption key, itself encrypted with the receiver's secret key, to the recipient.

To make sure that the sender receives a unique text encryption key from the central authentication server, and that the recipient and sender are totally confident that they are the actual and only persons in communication with each other, a unique code is appended to the protocol items. Used sequence numbers would have to be maintained by all parties using the network in order that intruders would be spotted immediately.

To make sure an intruder is not transmitting recently used keys, a critical time lapse would have to be acknowledged by all users of the network.

The central authentication server when activated, performs the following duties:

(1) Identifies the initiator of a message.

(2) Identifies the addressee, that is, the receiver.

(3) Generates a unique text encryption key.

(4) Produces the sender's and receiver's secret keys.

(5) Generates a unique code like a time stamp or sequence number.

The following is an explanation of the symbols that will be used in the forthcoming pages.

SYMBOLS	MEANING
X	- the originator of the communication
Y	- the receiver of the message
KX	- X's secret encryption key known only to him
KY	- Y's secret encryption key known only to him
TEK	- the text encryption key unique to each message
TS	- time stamp
CAS	- central authentication server

1. X --> CAS {X,Y}

X informs the CAS he wishes to communicate with Y. He identifies himself as well as the intended party he wishes to communicate with.

2. CAS --(encrypted)--> X

$$\{TEK,Y,TS1,\{TEK,X\}^{KY}\}^{KX}$$

The CAS retrieves X and Y's secret keys, generates a text encryption key that is associated only with this message, as well as a time stamp TS1. The CAS then sends all this back to X encrypting the item with X's secret key, KX. The time stamp is sent back along with the addressee information in order that X may verify that indeed it is the CAS responding and not an impostor. Since the item is encrypted with X's secret key, he is the only one that can decipher it. As a further step, X compares TS1 with the current time in order to determine if the time taken by the CAS to process the request is within the preset standard.

If (current time - TS1) > the standard, then communication ceases, and an investigation follows. Otherwise, step 3 is taken.

3. X --(encrypted)--> Y $\{X,TEK,TS2\}^{KY}$

Upon receipt by X, X decrypts the item in step 2 and sends the following to Y under Y's secret key, KY:

- X's id,
- the text encryption key
- another time stamp

4. Y --(encrypted)--> X $\{Y,TS3\}^{TEK}$

To prove to X that he, Y, has received the acknowledgement message of step 3, stating that X wishes to communicate with him, he returns a short time-stamped message to x encrypted with the TEK.

5. X --(encrypted)--> Y $\{X,\text{text of message}\}^{TEK}$

Upon receipt of the item in step 4, X compares the two time stamps TS2 and TS3. If their difference is greater than an agreed upon value by all participating members, then the item in step 4 may have been sent by an intruder. If that is the case, X informs Y that communication must cease at once, since all is not in order. If on the other hand everything checks out to be in order, then communication proceeds.

5.2.2 Multiple Channel Protocol

Another protocol that falls under the Conventional method, is one that involves more complex hardware.

Basically, each terminal and the central authentication server in the network is able to communicate over two channels. The one channel is the route the encrypted text would take. This channel is not as secure as the second channel. The second channel is used only for short protocol acknowledgement 'handshaking' messages. It could either be too costly and/or extremely slow for the text to be transmitted over this highly secured channel. Examples of such a channel are, the telephone, but using telephones that are not normally used by the sender and receiver, the mail, either registered or by private entrusted courier, hidden private circuit, and person-to-person encounter prior to transmission.

The following is a description of the protocol that takes place prior to text transmission. The items in steps 1 and 2 are made over the highly secured channel.

1. X --> CAS {X,Y,TS1}

X lets the central authentication server know he wishes to communicate with Y.

2. CAS --> X {X,Y,TEK,TS2}
   CAS --> Y {X,Y,TEK,TS2}

The CAS sends the same time stamp and text encryption key to both X and Y at the same time. Eventhough these transmissions need not be encrypted since the highly secure channel is being utilized, extra safety may be achieved if these transmissions are encrypted using both X and Y's secret keys.

X compares TS1 and TS2 and confirms whether the time difference falls within a preset network standard. If the standard is met, transmission proceeds. If not, all protocol operations cease, and an investigation is made on the longer than usual delay.

3. X --> Y $\{X,TS2,text......\}^{TEK}$

Transmission then commences with X sending the first block of text along with the time stamp TS2, all encrypted with the TEK. Y, knowing TEK, decrypts this first block of text, compares the time stamp originally received from the CAS, namely TS2, with the one received in the first block of text. If they are the same, Y accepts the transmission. Otherwise, an intruder is asssumed to be on the network and communication is not pursued any further.

5.2.3 Multiple Servers

One of the differences between this protocol method and that of using a single server, is that there is more than one authentication server per network 'protecting' the terminals. A central authentication server may exist, but it would act as an umbrella over the other servers in the network. This central authentication server would be mainly used to interface with other types of connecting networks. See D.K. Branstad {1} for more details.

Another difference between the single and multiple server networks, is cost. The network with the single server need only concern itself with the purchase, rental, or lease, of only one unit. Maintenance costs would be much less for the up keep of one unit instead of many. In the area of security, the multiple server network would come out on top. A sender and a receiver, no doubt, would be scrutinized by at least one authentication server, and possibly as many as three or four, all depending on the message's destination as well as the strategic positioning of the various servers throughout the network.

To identify which is better would be quite unfair. Each network has its own characteristics, requirements, security needs, and affordability. The higher the security needed for the messages of a network, the greater the number of authentication servers should be present to ward off unauthorized individuals.

In the description of the multiple server protocol that follows, we select the optimum network wherein, there exists one authentication server per terminal. Consequently, a message traverses through as many servers as there are terminals.

SYMBOL		MEANING
X	-	the originator of a message
Y	-	the receiver of the message. A ~~and B~~ are to communicate with each other in the example that follows.
AS	-	authentication server
ASX	-	X's authentication server
ASY	-	Y's " "
.		.
.		.
.		.
ASn	-	n's " "
TS	-	a unique time stamp
TEK	-	the text encryption key used to encrypt and decrypt the message
SKASn	-	n's authentication server's secret key
KX	-	X's secret key
KY	-	Y's secret key

1: X --> ASX {X,Y,TSI}

X informs his authentication server that he wishes to communicate with Y. This item is sent in clear text. X generates a unique time stamp associated only with this message.

2. ASX --(encrypted)--> ASn $\{X, Y, TEK, TS1\}$

The authentication server after generating a unique text encryption key, appends the codes of the intending communicants, as well as the time stamp received from the originator, encrypts these 4 items with its own encryption key, SKASA, and sends everything to the next authentication server, if one exists. If one does not exist, step 6 is taken. This step 2, informs the next server of the impending communication between X and Y.

3. ASn --(encrypted)--> ASY $\{X, Y, TEK, TS1\}^{SKASn}$

The request to communicate with Y filters from server to server until Y's authentication server is encountered. Each server is able to decrypt the items as it knows the secret encryption key of the surrounding servers. Also, each server has the power to authenticate the requests as far as the ID's, TEK, and the path taken by the transmission.

4. ASY --(encrypted)--> ASn $\{(TEK, A)^{KY}, X, Y, TEK, TS1\}^{SKASY}$

ASY acknowledges to the previous server that everything is in order by generating B's secret key, KY and sends the items back through the same servers to X.

5. ASn --(encrypted)--> ASX $\{(TEK, X)^{KY}, X, Y, TEK, TS1\}^{SKASn}$

Included in the items being sent back to the originator, is the item that X will send to Y as his initial approach to Y, that is, $(TEK,X)^{KY}$.

6. ASX --(encrypted)--> X $\{(TEK,X)^{KY},X,Y,TEK,TS1\}^{KX}$

ASX sends back to X virtually the same item it received from the previous servers, except that the items are now encrypted with X's secret key KX. X is the only one that can decipher these items.

7. X --(encrypted)--> Y $\{TEK,X\}^{KY}$,{TS2}

X lets Y know he wishes to communicate with him by sending his, namely X's, identity along with the text encryption key to use. X generates a second time stamp TS2. X compares TS2 with TS1 to determine whether standards have been met. Only Y can decrypt the transmitted items. After decrypting the first item, he finds TEK which he will use on the transmissions he will receive from X.

8. Y --> X $\{TS2\}^{TEK}$

As an acknowledgement to X that he did contact him, Y encrypts TS2 with TEK and sends it back to to X.

9. X --(encrypted)--> Y $\{text......\}^{TEK}$

If all is in order, and both X and Y are assured of each other's identity, and that no one else is 'listening in' on their conversation, text transmission begins using the text encryption key to encrypt the text. We have involved ourselves mainly in the encrypting functions. It must be noted that a key is also necessary to decrypt what has been encrypted. Knowing the TEK, the decryption key is automatically known under the Conventional Protocol system.

5.3 Public Key Protocol

In a Public Key encryption system, two keys are necessary. One is used to convert clear text to cipher text, while the second is used to convert back from cipher-text to clear text. Knowledge of one key does not help in finding the other, and the two keys will act as inverses for each other. A message encrypted by X's public key can only be decrypted by using X's secret key.

5.3.1 Single Server

As in section 5.2.1, we are dealing with a single central authentication server as opposed to many.

SYMBOL		MEANING
X	-	message originator
Y	-	message receiver
CAS	-	central authentication server
SKCAS	-	the cas's secret key
PKX	-	X's public key
PKY	-	Y's public key

1. X --> CAS {X,Y}

X consults the central authentication server, CAS, in clear text, to find Y's public key PKY, as he wishes to communicate with him.

2. CAS --(encrypted)--> X $\{PKY,Y\}^{SKCAS}$

The CAS replies to X using its secret key, SKCAS. X must know the CAS's public key, having obtained it previously in a reliable way. The importance of the reciprocity between the public and secret keys is shown here. The encryption of the message in Step 2 is required, not to ensure the privacy of the information, but to ensure its integrity. X must be sure that PKY is the correct public key of his intended receiver Y, and not of someone else. X knows that the identity of his intended receiver Y was properly communicated to the CAS in the message of Step 1, since it was properly returned in the message of Step 2.

3. X --(encrypted)--> Y $\{TS1,X\}^{PKY}$

Communication with Y is then initiated. This message can only be decrypted by Y using his secret key SKY. Besides the identifier X, of the sender, X also includes TS1 a unique time stamp or sequence number.

4. Y --> CAS $\{Y,X\}$

CAS --> Y $\{PKX,X\}^{SKCAS}$

Now, Y finds the originator's public key using the same procedure as in Steps 1 and 2.

A double handshake is now needed to authenticate the sender and receiver to one another.

5. Y --(encrypted)--> X $\{TS1,TS2\}^{PKX}$

Y returns the unique time stamp X sent him in Step 3, along with another unique code generated by himself.

6. X --(encrypted)--> Y $\{TS2\}^{PKY}$

The handshake is complete by X returning the unique code that was generated and sent by Y using Y's public key, PKY.

7. X --(encrypted)--> Y $\{text.....\}^{PKY}$

Text communication may now begin, X and Y being assured they are transmitting to each other and not to an impostor.

5.3.2 Multiple Servers

This section resembles section 5.2.3 as far as the network design is concerned. There are many servers rather than only one. The following is a step by step description of public key protocol when multiple servers are present in the network.

SYMBOL		MEANING
PKX	-	public key of X
PKY	-	public key of Y
SKX	-	secret key of X
SKY	-	secret key of Y

SYMBOL		MEANING
ASX	-	authentication server for X
ASY	-	authentication server for Y
⋮		⋮
ASn	-	the nth authentication server
SKASX	-	secret key of auth. server X
SKASY	-	secret key of auth. server Y
⋮		⋮
SKASn	-	secret key of the nth auth. server
TS	-	a unique time stamp

1. X --> ASX $\{X,Y\}$

X lets his authentication server, ASX, know he wishes to communicate with Y. This is done in clear text.

2. ASX --(encrypted)--> ASn $\{X,Y,PKX\}^{SKASX}$

The message filters from server to server, until the server associated with the receiver is encountered.

3. ASn --(encrypted)--> ASY $\{X,Y,PKX\}^{SKASn}$

ASX lets ASY know that X wishes to communicate with Y. Provides X, and PKX as proof. ASn requests Y's public key, PKY from ASY. This is all encrypted with next to last authentication server's secret key, SKASn

4. ASY--(encrypted)--> ASn $\{X,Y,PKY\}^{SKASY}$

ASY supplies Y's public key since all information supplied is correct and no suspicion is raised.

5. ASn--(encrypted)--> ASX $\{X,Y,PKY\}^{SKASn}$

The information required by X finally arrives to X's server.

6. ASX--(encrypted)--> X $\{X,Y,PKY\}^{SKASX}$

As soon as [illegible] is received by ASX, it is sent on to X.

7. X --(encrypted)--> Y $\{X,TS1\}^{PKY}$

X uses Y's public key to encrypt the item that will let Y know that he wishes to communicate with him.

Y goes through the same gyrations to obtain X's public key from ASX.

8. Y --> ASY $\{X,Y\}$

9. ASY --(encrypted)--> ASn $\{Y,X,PKY\}^{SKASY}$

10. ASn --(encrypted)--> ASX $\{Y,X,PKY\}^{SKASn}$

11. ASX --(encrypted)--> ASn $\{Y,X,PKX\}^{SKASX}$

12. ASn --(encrypted)--> ASY $\{Y,X,PKX\}^{SKASn}$

13. ASY --(encrypted)--> Y $\{Y,X,PKX\}^{SKASY}$

Finally, when the public keys of each other are known, a short handshake is all that is needed before actual transmission may commence.

14. Y --(encrypted)--> X $\{TS1,TS2\}^{PKX}$

Y generates a second time stamp and along with the one he originally received from X, encrypts them with X's public key and sends the item to X.

15. X --(encrypted)--> Y $\{TS2\}^{PKY}$

The final protocol message has X returning Y's generated time stamp as a last confirmation of their identity.

16. X --(encrypted)--> Y $\{text.......\}^{PKY}$

X now sends the text encrypted with Y's public key. Y uses his secret key known only to him to decrypt it.

5.4 Differences Betwoon Conventional and Public Key Protocol Methods

The main differences in these two methods of pre-text transmission handshaking are the number and type of keys needed to do the job. The Conventional method requires one key to encrypt and decrypt, and it is secret, known only to those concerned. The Public Key method requires two keys. One to encrypt, and another to decrypt. The public key is known to anyone who cares to know it. But, once encryption takes place using this public key, it is only the holder of the corresponding secret key that can decrypt that same message back to its clear text.

5.5 Protocol Time Delay

A study was undertaken to determine the effect the number of authentication servers would have on the amount of time taken to complete the protocol prior to actual text transmission. The study was conducted for the Conventional and Public Key protocol systems.

Time delays due to the protocols were analyzed. Our results follow.

The following figures depict time in CPU UNITS. A CPU UNIT is equal to approximately 26.04 micro seconds. The chart below shows how much time was used by the CPU to complete the protocol function just prior to the start of message transmission.

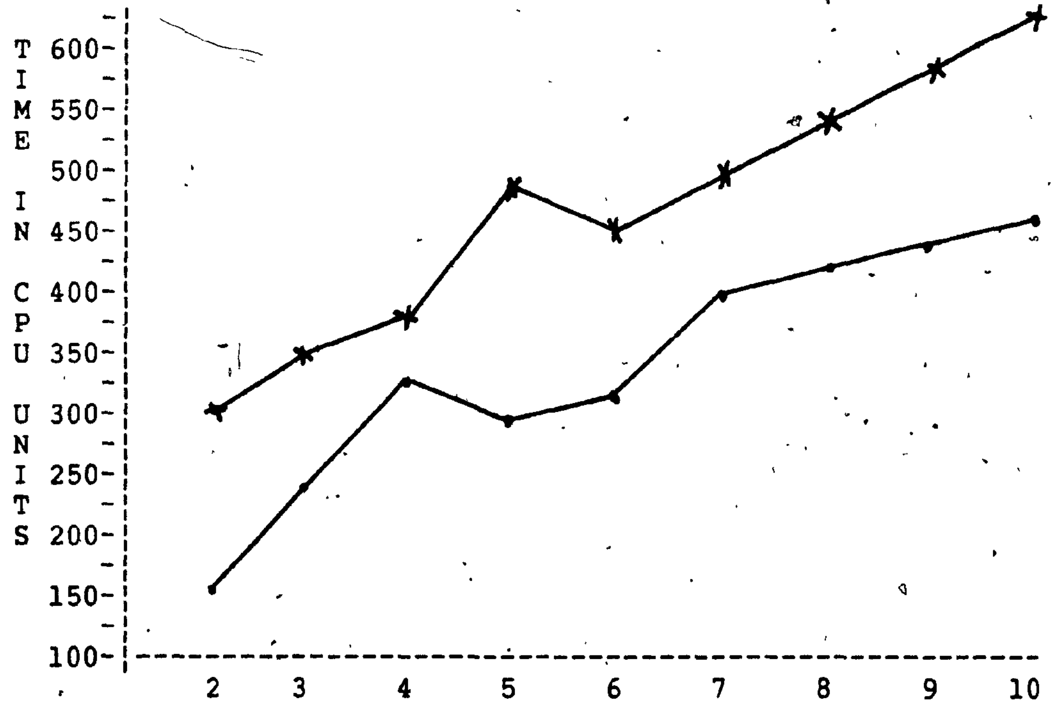
TABLE 5.1

PROTOCOL COMPLETION TIME (CPU UNITS)

	No. of Authentication Servers								
Protocol Method	2	3	4	5	6	7	8	9	10
Conventional(.)	156	239	325	287	313	391	421	441	465
Public Key(x)	305	343	377	481	451	489	533	569	605
Slope									
- Conventional	-	83	86	-	26	78	30	20	24
- Public Key	-	38	34	104	-	38	44	36	-

The reason for the missing slope values in Table 5.1, is that, for some unexplainable reason, the CLOCK routine returned a smaller time value for an increase in the number of servers. For instance, 5 servers expended 481 CPU units, while 6 servers took 451 CPU units to complete their task. No explanation can be given for this phenomenon.

CONVENTIONAL AND PUBLIC KEY PROTOCOL TIME versus NUMBER OF AUTHENTICATION SERVERS



NO. OF AUTHENTICATION SERVERS BETWEEN SENDER/RECEIVER

To obtain the processing time for this analysis, we had to write a module using the assembler language, since IBM's FORTRAN IV does not have this ability. The module is called CLOCK and may be found in the appendix of this thesis. It was link edited along with the fortran MAIN program just prior to execution. The macros TIMER and STIMER were executed at the appropriate places by the fortran program to compute the processing time.

The anomalies at the 5 server mark are not quite explainable, since all authentication servers are treated the same in the program.

From the preceeding graphs, and the slope of the curve values, it may be deduced that the relationship of protocol processing time and number of servers is not linear.

As expected, the more authentication servers between the sender and receiver, the longer it takes for the protocol function to finish. It could also be expected that, message traffic will be slowed as it passes through the various servers. Consequently, the message through-put at these 'potential bottle necks', is one more parameter that must be considered in network design. Finally, as there are close to twice as many steps for the Public Key protocol function, namely 16 steps versus 9, it was expected to take longer than the Conventional system. From Table 5.1, the Public Key protocol system takes between 1.3 to 1.7 times longer than the Conventional system.

CHAPTER 6

DIGITALIZED SIGNATURES AND MESSAGE ENCRYPTION

6.1 Preamble

In practically all business transactions, an essential role is played by signed messages and by certification of messages received. The signature, which is assumed to be unique to the signatory or signer, should serve as proof that the originator was a party to the document, or that he was its sole originator.

W.E. Ulrich {19}, {20}, discusses the movement towards the era of electronic mail. A larger portion of correspondence between departments of the same or different companies will be done using teleprocessing equipment. The most widey used terminal at present is the cathode ray tube equipped with a typewriter keyboard.

When corresponding in this mode, there arise problems of how to affix a binding signature to a message when this is deemed necessary, and the message with the signature be accepted as being created only by the signator without any repudiation whatsoever.

There must be no doubt at all in the receiver's mind who the originator is, otherwise there seems to be no sense in taking part in a communication network where classified information is being transmitted. G.J. Popek {16} beleives that the function of authentication is and will become so important that, 'unforgeable' mechanisms to take and validate fingerprints, or other personal characteristics, will soon emerge.

The signature as well as the text of the message is just information, that is, a string of bits devoid of any physical characteristics.

The first part of this chapter is devoted to digitalized signatures, while the second part is involved with some methods in message text encryption. The encoding function analyzed in this section, eventhough used in digitalizing signatures, could also be used in message text encryption. Slight modifications may need to be made to the function itself in order to help in decrypting the cipher text.

6.2 Digitalized Signatures

Michael O. Rabin {27}, proposes "a signature system employing any block-encoding device and based, in one essential aspect, on probabilistic logic".

The message is denoted by M.

The signature on M by a person P is denoted by $S_P(M)$.

The properties of the signature are:

(a) Only P can produce any pair $(M, S_P(M))$,

(b) The recipient of a pair (M,W), claimed to be signed by P, can check that indeed $W = S_P(M)$. Property (a) entails that the signature is not only characteristic of the signator P, but also of the entire message M. If when given a signed message $N, S_P(N)$ an adversary could effectively find a message M that is not equal to N such that,

$$S_P(M) = S_P(N),$$

then the adversary could produce a signed message $(M, S_P(M))$ not authorized by P. This contradicts (a). Ordinary signatures do not enjoy this important property of never changing or varying.

6.3 Encoding

The theoretical encoding function is as follows:

Encoding function set is a mapping E,

$$\{0,1\}^k * \{0,1\}^k \dashrightarrow \{0,1\}^k$$

For x and w included in $\{0,1\}^k$, we denoted the function value $E_x w$ and call it the encryption of w with the key x.

The message is converted to numeric form by straight replacement using the index of each character in the alphabet. For example, A = 01, B = 02, etc.

The word length $l(w) = 2$ characters c_1 and c_2.

The key length was also selected as 2 characters, that is, equal in length to one word. As in the case of the characters of a message, the keys were also converted to numeric form using the same relationship.

In our first attempt to come up with an encryption function using the key to encypher a word, we encountered many problems. The results obtained in most of the cases was an encrypted word with a value of zero. The mathematical relationship between a key and a word it is to encrypt, is the same relationship a word, used as a key, has on another word that it is to encrypt. Also, the same relationship holds when an entire message is used as a key to encrypt a word. The mathematical operation of the key x_i on the word w_i in $E_{x_i} w_i$ is the same in the function w_i on the word w_j in $E_{w_j} w$ when computing the function:

$$E_M w = E_{w_1} (E_{w_2} \ldots\ldots\ldots (E_{w_{M-1}} (E_{w_M} (w))) \ldots\ldots\ldots)$$

In this last relationship, the entire message M is used as a key. The mathematical operations for the 1st function attempted were as follows:

$E_x w$ had a key x made up of two characters c_{x_1}, and c_{x_2}. The word 'w' also consisted of two characters. These were denoted by c_{w_1} and c_{w_2}. The key x was either a randomly generated 2-character key, or a word from the message text itself. Encoding proceeded from right to left when $E_M w$ was computed. That is, the innermost computation was done first. Once $E_{w_M}(w)$ was computed, say = RESULT, then $E_{w_{M-1}}$ (RESULT) was calculated, and so on, moving from right to left.

The mathematical operation was as follows:

$$E_x(w) = c_{x_1} c_{x_2} (c_{w_1} c_{w_2})$$

$$= (|c_{x_1} - c_{w_1}| * w) \bmod 29, \quad (|c_{x_2} - c_{w_2}| * w) \bmod 29$$

The number 29 was chosen since it is the first prime number following the numeric representation of the character 'z', namely 26.

For example, the word YOUb, where 'b' denotes a blank and is equal to 00, translates numerically to:

Y	O	U	b
25	15	21	00

If we now use the first two characters as the key to encrypt the last two characters, we obtain:

$$E_{YO}(Ub) = E_{25\,15}(21\,00)$$

$$= (|25-21| * 2100) \bmod 29, \quad (|15-00| * 2100) \bmod 29$$

$$= 8400 \bmod 29, \; 21500 \bmod 29$$

$$= 19 \quad , \quad 06 \quad = 1906$$

As stated before, our first attempt at an encoding function proved inadequate. When it was used to compute the values of different compressed messages, described in later sections of this chapter, the values were almost always zero. The results were the same even after trying different modulo primes. The reasons attributed to these disastrous results were two-fold.

1. zero resulted frequently in either of the two subtraction operations,

2. the numeric value of w in $E_x(w)$ very often was an exact multiple of the modulus used. The second encoding function proved to be much more stable.

Our second attempt at producing an encoding function was as follows.

The main differences between the two versions of the encoding function are:

1. powers are used instead of multiplication, and,

2. logic is added to prevent results of zero in the subtraction operations.

Let $E_x(w) = c_{x_1} c_{x_2} (c_{w_1} c_{w_2})$.

If $c_{x_1} \neq c_{w_1}$

then calculate $((|c_{x_1} - c_{w_1}|)^{w_1 \bmod 7}) \bmod 29$

else if $c_{x_1} \neq c_{x_2}$

then calculate $((|c_{x_1} - c_{x_2}|)^{w_2 \bmod 7}) \bmod 29$

else calculate $c_{x_1}^{w_1 \bmod 7} \bmod 29$.

If $c_{x_2} \neq c_{w_2}$

then calculate $((|c_{x_2} - c_{w_2}|)^{w_2 \bmod 7}) \bmod 29$

else If $c_{w_1} \neq c_{w_2}$.

then calculate $((|c_{w_1} - c_{w_2}|)^{w_2 \bmod 7}) \bmod 29$

else calculate $c_{x_1}^{w_1 \bmod 7} \bmod 29$.

The modulus of 7 was used in the exponent in order to reduce the magnitude of the result, and consequently make it more manageable.

6.4 The Standard Message M_0

The Standard Message is a bit string of length k, that must be used by everyone using the system,

$$M_0(i) = 0^{k-e} E_{e-1} \ldots\ldots\ldots\ldots E_0 .$$

If i = 5, $M_0(5) = 0^{k-3}101$ and $k = l(M_0(i))$ which is the length of one word. In our case, one word = 2 characters = 16 bits

$M_0(i)$ was first contemplated. This standard message is used in conjunction with S, the number of keys, to be discussed in later sections of this chapter. The first n bits required to accommodate the maximum value of S in binary in $M_0(i)$ were initially made zero. If they are not zero, duplicates ensue.

For example, if $M_0 = 101110_2 = 46_{10}$,

then $M_0(1) = 101111 = 47$,

$M_0(2) = 101110 = 46$ and $M_0 = M_0(2) = 46$.

If we now take S in this example as equal to a total of 4 keys, then 4 in binary = 100 and requires 3 bit positions. It is these first 3 bit positions that are initially made equal to zero so that with every increase in 'i' of $M_0(i)$, a unique value will result.

$M_0(i)$ can then take on the unique values of:

$M_0 = 101000 = 40$, $M_0(1) = 101001 = 41$,

where, the first 3 bits are arbitrarily chosen and the last 3, initially set to zeroes, accommodate the maximum value of 4.

$M_0(2) = 101010 = 42$ etc....

and no duplicates occur.

The value for M_0 for this analysis was arbitrarily taken as 0001011000000000 = 5632. Both, the above mentioned constraint to avoid duplicates, and the total number of keys were taken into consideration when the value of M_0 was chosen. Then, $M_0(1) = 5633$, $M_0(2) = 5634$, ...etc.

6.5 The Compressed Message C(M)

Compressing the message is one of the functions a sender must perform. The compressed message is defined as:

$$C(M) = E_M M_0$$

That is, the standard message, M_0 is encrypted using the entire message M as a key. In computing C(M), E_M is first calculated and then used as the key and encoded onto M_0. An example follows:

EXAMPLE 6.1

The message M = I T b W A S

Numerically M = 09 20 00 23 01 19

Proceeding from right to left, and using the encryption function of our second attempt, we have:

$$\text{Step 1. } E_{(0023)}(0119) = (|00-01|^{01 \bmod 7}) \bmod 29, \quad (|23-19|^{19 \bmod 7}) \bmod 29$$

$$= 01 \bmod 29 \quad , \quad 1024 \bmod 29$$

$$= 0109$$

$$\text{Step 2. } E_{(0920)}(0109) = (|09-01|^{01 \bmod 7}) \bmod 29, \quad (|20-09|^{09 \bmod 7}) \bmod 29$$

$$= 08 \bmod 29 \quad , \quad 121 \bmod 29$$

$$= 08 \quad , \quad 05$$

$$E_M = 0805$$

Step 3. Now, $C(M) = E_M(M_0)$, where $M_0 = 5632$

Therefore $C(M) = E_{(0805)}(5632) = 0116$

6.6 Key Selection

If A and B wish to conduct digitalized signed correspondence, they must select a number of keys at random which they do not divulge to each other at this time.

The parameter S represents the number of keys selected.

A and B chose S keys each:

$$A = x_1, x_2, x_3, \ldots\ldots, x_S$$

$$B = y_1, y_2, y_3, \ldots\ldots, y_S$$

These keys are chosen at a time when A and B are together. At ~~the same~~ time the keys are chosen, the following two ordered lists, a and b are computed using the encoding function.

For A, LIST $a = E_{x_1}(M_0(1)), E_{x_2}(M_0(2)) \ldots\ldots\ldots, E_{x_S}(M_0(S))$

For B, LIST $b = E_{y_1}(M_0(1)), E_{y_2}(M_0(2)) \ldots\ldots\ldots, E_{y_S}(M_0(S))$

A and B then sign, by ordinary legal procedure, an agreement stating that 'a' is an encoding of the Standard Message using A's keys, and 'b' is an encoding of the Standard Message using B's keys. It is then possible for B or any other party to be presented with a word x claimed to be A's i th. key, x_i. A can then verify that the word is indeed his x_i key by computing $E_{x_i}(M_0(i))$, and comparing it with a_i in the list a. Same verification is possible for B, or for that matter anyone else that is part of the system.

6.7 The Production of Signatures

After compressing the message M, the sender performs his second function, that of calculating his signature.

The parameter T represents a block of keys and is a subset of the S keys originally selected.

The sender selects the first block of T keys from his bag of S keys. Each message is signed with a different block of keys, to avoid discovery through repetition. For this reason, the S keys chosen are divided into sets of T, and each message sent uses the next block of keys.

The signature is defined as a list of markings.

The signature of A on a message M is:

$$S_A(M) = E_{x_1} C(M), E_{x_2} C(M) \ldots\ldots\ldots\ldots, E_{x_T} C(M)$$

where, each $E_{x_i} C(M)$ is a marking.

The number of markings, T, is arbitrary, but in an actual implementation, it all depends on how secure one wishes the system to be. A then appends the signature to the message M and sends it to B.

The following is the procedure for validation of a signature.

When B receives the message-signature sequence, $M, (u_1, u_2, \ldots\ldots\ldots, u_T)$ from A, he verifies that indeed,

$$(u_1, u_2, \ldots\ldots\ldots\ldots u_T) = S_A(M)$$

by the following procedure.

The parameter R represents the random key indices selected.

Step 1. B randomly choses R different numbers, where

$$1 <= i_j <= T \quad \text{and} \quad 1 <= j <= R$$

Step 2. Upon request from the receiver B, the sender A divulges to him the actual keys whose indices he randomly chose in step 1, $x_{i_1}, x_{i_2}, \ldots\ldots\ldots\ldots\ldots, x_{i_R}$

Step 3. B verifies that $x_{i_1}, x_{i_2}, \ldots\ldots\ldots\ldots\ldots\ldots, x_{i_R}$ are indeed A's i_1 th.,......, i_R th. keys by computing:

$$E_{x_i}(M_0(i)) = a_i ,$$

and compares the R above results with the a_i entries in LIST a.

Step 4. Once the keys are verified and are found to be authentic, B then checks that R of the T markings received as part of the signature do also belong to the sender:

$$u_{i_j} = E_{x_{i_j}}(C(M)) \quad \text{where} \quad 1 <= j <= R$$

That is, these results would be compared with those sent along with the signature. The receiver B would then accept:

(1) The signature as being that of sender A.

(2) The message as originating from A, if and only if all the tests in steps 3 and 4 resulted positively.

6.8 Adjudication of Disputes

From time to time, a sender may want to challenge or disown a message claimed to be signed by him. The receiver would then present to the adjudicator, a (message, markings) set, that is, $(M, v_1, v_2, v_3, \ldots\ldots\ldots, v_T)$, claiming that it does belong to the disclaiming sender.

The adjudicator requests the sender to reveal the keys he allegedly employed to produce the markings making up the signature.

(i) The adjudicator proceeds to verify that:

$$E_{x_1}(M_0(1)) = a_1, \ldots\ldots\ldots\ldots E_{x_T}(M_0(T)) = a_T .$$

If not all keys are verified, the adjudicator right away upholds the receiver's claim that the signature is valid.

(ii) After all keys are verified, each marking of the signature received is tested to see if $E_{x_i}(C(M)) = v_i$.

Step 3. If fewer than a certain number of these equalities are true, then the judge upholds the sender's claim that, indeed the message was not signed by him. If on the other hand more than a certain number of equalities are true, then the balance sways in favour of the receiver, that is, the sender did indeed sign and send the message.

The situation may arise wherein a sender wishes a receiver to accept a message, but one that is not signed or sent by him, that is, the sender. The only way for this originator to produce a seemingly signed message, such as:

$$(M, u_1, u_2, \ldots\ldots\ldots, u_T),$$

which the receiver will accept and which can later be successfully denied by the originator, is for the receiver in the acceptance stage to produce exactly T markings, $u_{j_1} .. u_{j_T}$ which are proper, that is, $u_{j_i} = E_x(C(M))$. Here we assume the receiver was told by the sender that a certain block of keys was used. Exactly T, or more markings must be proper, for if fewer, the message will not be accepted, by Step 4 of the signature validation procedure of Section 6.7. If more than T markings are proper, then because of Step 3 of the Adjudication of Disputes procedure of Section 6.8, the sender cannot successfully challenge this signed message.

If the sender has prepared the (message,signature) set $(M, u_1, u_2, \ldots\ldots, u_{40})$ and that a random R = 20 of the 40 keys are selected for validating purposes by the receiver, the latter would have to pick, in his random selecting, exactly the indices $j_1, j_2, \ldots\ldots j_{20}$. The probability of this occurring is

$$\frac{1}{\binom{40}{20}}.$$

Thus, the receiver can be cheated on the average no more than once in 10^{11} times that he accepts a signed message when he shouldn't have.

6.9 A Study of the Relationship of Processing Time versus Message Length for Different S, T, and R Parameter Values

A study was conducted to determine the relationship between the message length, l(m), and processing time. This test was carried out for different combinations of the three parameters discussed in this chapter, namely S, T, and R. The processing time included:

(1) Computing the compressed message C(M).

(2) Generating the signature $S_A(M)$ with all the T markings.

(3) Authentication of the markings.

The results of our study were as follows.

We wanted to determine whether there was a relationship between the processing time and the length of the message processed, for certain values of S, T, and R. Also, for particular processing times and message lengths, we wanted to observe the above relationship at different S, T, and R values.

In the analysis, the message length was initially taken to be 80 characters. Each succeeding run had a message length 4 times the previous length, until a maximum of 81920 characters per message was reached. The values of the parameters were taken such that:

$$20 <= S =< 200$$

$$.25S <= T =< .33S$$

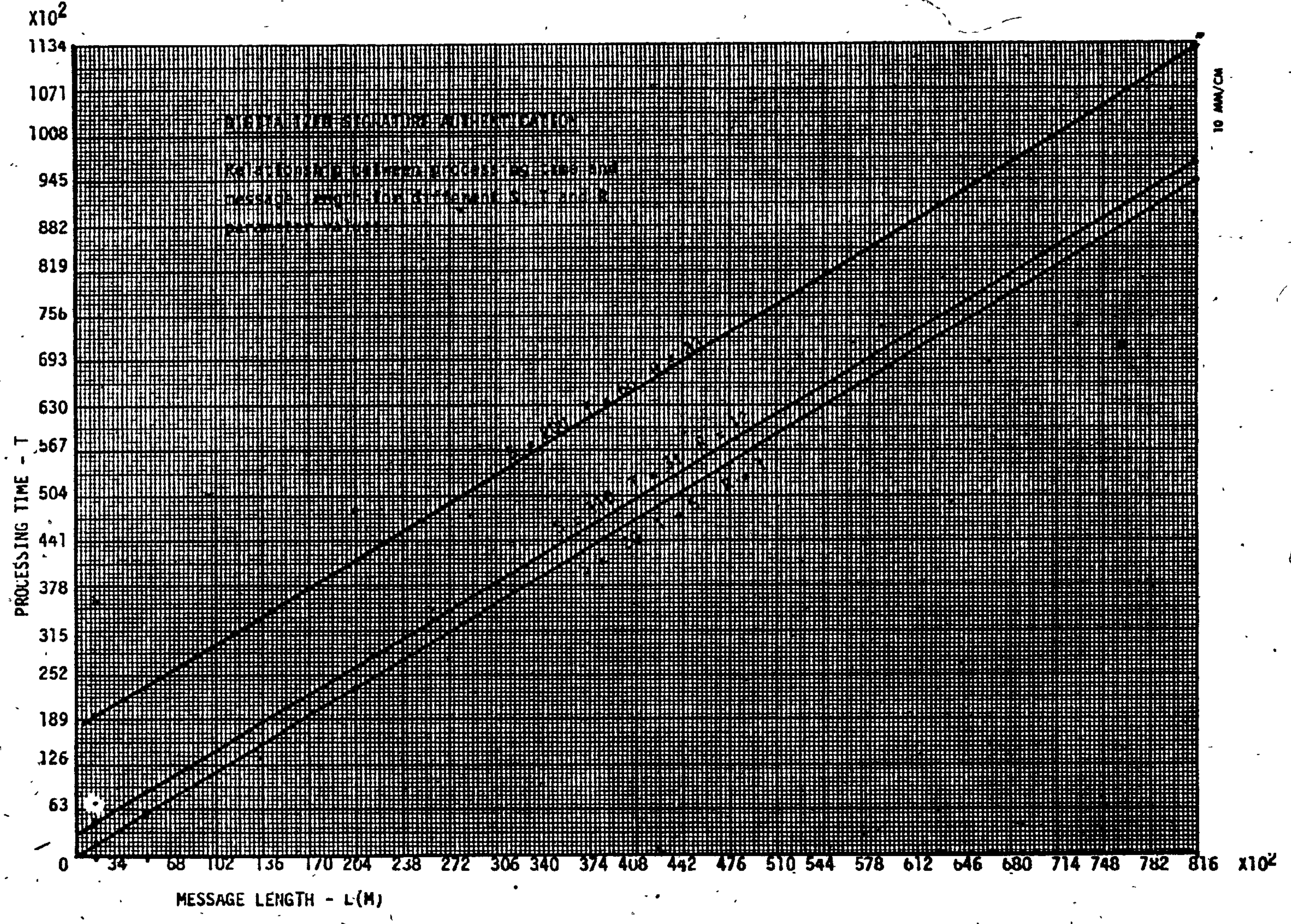
$$.25T <= R <= .50T$$

The CPU UNITS mentioned in Table 6.1 are the same time units as described in Chapter 5.

TABLE 6.1

Message Length (chars) L(M)	Total No. Keys S	Number of Keys Chosen T	Number of Random Key Indices Chosen R	Processing Time in CPU Units U
80	20	5	1	227
320	20	5	1	499
1280	20	5	1	1613
5120	20	5	1	6045
20480	20	5	1	23729
81920	20	5	1	94661
80	20	6	1	223
320	20	6	1	499
1280	20	6	1	1613
5120	20	6	1	6027
20480	20	6	1	23845
81920	20	6	1	94547
80	20	6	2	251
320	20	6	2	521
1280	20	6	2	1629
5120	20	6	2	6055
20480	20	6	2	23763
81920	20	6	2	94629
80	110	26	6	2421
320	110	26	6	2711
1280	110	26	6	3777
5120	110	26	6	8199
20480	110	26	6	25993
81920	110	26	6	97043
80	110	31	12	2447
320	110	31	12	2721
1280	110	31	12	3847
5120	110	31	12	8273
20480	110	31	12	26109
81920	110	31	12	97527
80	110	36	18	2535
320	110	36	18	2781
1280	110	36	18	3893
5120	110	36	18	8341
20480	110	36	18	26133
81920	110	36	18	97099

80	200	50	12	18233
320	200	50	12	18445
1280	200	50	12	19615
5120	200	50	12	24005
20480	200	50	12	41837
81920	200	50	12	113099
80	200	60	12	18113
320	200	60	12	18403
1280	200	60	12	19577
5120	200	60	12	24073
20480	200	60	12	41915
81920	200	60	12	112987
80	200	66	33	18395
320	200	66	33	18685
1280	200	66	33	19797
5120	200	66	33	24411
20480	200	66	33	42079
81920	200	66	33	113285


DIGITALIZED SIGNATURE AUTHENTICATION
Relationship between processing time and message length for different S, T and R parameter values.
PROCESSING TIME - T
X10²
0
63
126
189
252
315
378
441
504
567
630
693
756
819
882
945
1008
1071
1134
MESSAGE LENGTH - L(M)
34
68
102
136
170
204
238
272
306
340
374
408
442
476
510
544
578
612
646
680
714
748
782
816
X10²
10 MM/CM

The graphical representation on the preceding page, was made using nine sets of the output data from Table 6.1. The selection of the sets was totally arbitrary. The samples chosen are typical of the population, and are not biased in any way. From the graphs, it was observed that the relationship between the processing time and the length of a message for different values of the parameters S, T, and R, is linear. One 'curve' from each of the three sets was chosen for a mathematical verification of its linearity.

1. $S = 20,\ T = 5,\ R = 1$

$$\text{slope } m_1 = \frac{23729 - 227}{20480 - 80} = 1.152$$

$$\text{slope } m_2 = \frac{94661 - 6045}{81920 - 5120} = 1.154$$

2. $S = 110,\ T = 31,\ R = 12$

$$\text{slope } m_1 = \frac{26109 - 2447}{20480 - 80} = 1.160$$

$$\text{slope } m_2 = \frac{97527 - 8273}{81920 - 5120} = 1.162$$

3. $S = 200,\ T = 66,\ R = 33$

$$\text{slope } m_1 = \frac{42079 - 19797}{20480 - 80} = 1.092$$

$$\text{slope } m_2 = \frac{113285 - 24411}{81920 - 5120} = 1.157$$

Within reasonable error, the slopes m_1 and m_2 are approximately equal in each of the three sets. This signifies linearity.

TABLE 6.2

PARAMETERS			PROCESSING TIME/CHAR.
S	T	R	CPU UNITS/CHAR.
20	5	1	1.153885630
20	6	1	1.152541544
20	6	2	1.153201369
110	26	6	1.156182796
110	31	12	1.161779081
110	36	18	1.155474096
200	50	12	1.159164223
200	60	12	1.159261975
200	66	33	1.159457478

As expected, the processing time increased as the values of the parameters S, T, and R increased. From Table 6.2, it may be seen that the increase is not mutch.

Increase = (1.159457478 - 1.153885630) CPU UNITS/CHAR.

= 0.005578572848 CPU UNITS/CHAR.

= 0.005578572848 * 26.04 MICRO SECS./CHAR.

= $0.145\ldots \times 10^{-6}$ SECS./CHAR.

This small increase in processing time is a minute price to pay for a substantial increase in security. It is not worthwhile sacrificing better security and peace of mind, by having more keys for the few extra micro seconds needed to process the few extra keys.

6.10 Message Encryption

Encryption is the standard means of rendering a communication private. The sender enciphers each message before transmitting it to his intended receiver(s). The receiver, and no unauthorized person knows the appropriate deciphering function to apply to the received message to obtain the original message. An eavesdropper who 'hears' the transmitted message through some means of a wiretap, hears only the scrambled cipher text, which should make no sense to him since he does not know how to decipher it.

Not only is eavesdropping a danger, but actual transmitting by an intruder masquerading as an authentic network participant, should be anticipated. The intruder wants to appear authentic so that he could receive the valuable information. Duplicate messages must be sifted out by the authentic participants, for they may be the initial penetration by the intruder. Messages are usually transmitted in blocks. Each block should be treated separately, regarding it as a transmission in itself. To ensure that parts of messages are not stolen or destroyed, whether inadvertently, or not, G.J. Popek {16} mentions the block chaining method for ensuring continuity in a message. This procedure entails that a small segment of the preceeding encrypted block is appended to the current clear text block before encryption and transmission. The receiver can therefore easily check that blocks have been received in their proper order by making the proper checks. Including sequence numbers in each block, the receiver could determine the number of blocks lost.

In many instances, the deciphering function may be done away with for short, or even long messages. The sender and receiver would have had to come into secret person-to-person contact, or use a 100% secure communication link, prior to transmitting any text. The two parties would agree on the meaning of different cipher texts sent and received. This would therefore eliminate the step(s) for encrypting and decrypting texts. Shortening the handshaking session is a definite asset, but this method would still not offer a total resistance to unauthorized penetration.

The method of text encryption studied and presented here, is outlined by Messrs. R. I. Rivest, A. Shamir, and A. Adleman {17}. A description and example follow.

6.11 An Efficient Encryption Method Using Powers and Quotient Remainders

Given a message in clear text. The requirement is to come up with some encryption function that has a one to one relationship between the clear text 'item' and its encoded result. That is, if:

E denotes an encoding function,

x represents the key used in the encoding function, and

w_i is the i th. word or message block,

then, the result of encrypting w_i and w_j, where w_i and w_j are two different items, using the same encryption function and key on both, must not produce the same results.

That is, $E_x(w_i) \neq E_x(w_j)$, and $w_i \neq w_j$

The analysis of such a system follows.

The key is a pair of positive integers (e,n), to be described later. Each 'item', or in this case letter of the text, is converted to a unique number. The message text is broken up into blocks. In our analysis, a block consisted of 2 characters. Each message block is encrypted by raising it to the power e (modulo n). That is, the cipher text is the remainder when the block B is raised to the e th power, and the result divided by n.

If C is the resulting cipher text,

$$C = E(B) = B^e \pmod{n}$$

'n' is the product of two large prime numbers, p and q. Their magnitude ranges in the one hundred digit area. In our example, to simplify the calculations, but without any theoretical loss, we chose p and q to be small prime numbers, namely p = 3 and q = 5.

$$n = p * q \qquad (1)$$

This method falls into the category of Public Key Encryption. The reason being that, the values e,n are made public. The values of p and q on the other hand, are kept secret. Not knowing these factors of n, an intruder would find it quite difficult to calculate e and d. The parameter 'd' is discussed below. The larger the prime values of p and q, the more difficult it is to resolve the cipher text into the clear text.

The decoding function is similar, except that the power used on the ciypher text is different,

$$D(C) = C^{d} \pmod{n}$$

'd' is relatively prime to the product

$$(p-1) * (q-1).$$

That is, the gcd(d,((p-1) * (q-1))) = 1 (2)

The value e in the encoding function is the multiplicative inverse of d mod ((p-1) * (q-1)), that is,

$$(e) * (d) = 1 \pmod{((p-1) * (q-1))}$$

PHI(n) represents the Euler Totient. This is defined as the number of coprimes that are less than a certain value.

For example, PHI(15) = number of coprime values of $x < 15$.

Coprime values of 15 = 1,2,4,7,8,11,13,14.

That is, PHI(15) = 8.

Now, PHI(n) = (p-1), where p is a prime number. For a number n = (p) * (q), where p and q are primes, and by the elementary properties of the totient function:

$$\begin{aligned} PHI(n) &= (PHI(p) * (PHI(q)) \\ &= (p-1) * (q-1) = pq - p - q + 1 \\ &\qquad = n - (p + q) + 1 \\ &\qquad = 15 - (3 + 5) + 1 = 8 \text{ as} \end{aligned}$$

above.

Since d of the decryption function is relatively prime to PHI(n), it has a multiplicative inverse e in the ring of integers modulo PHI(n):

$$(e) * (d) = 1 \pmod{PHI(n)} \qquad (3)$$

To compute e, use is made of an extension to Euclid's algorithm for computing the greatest common divisor. The integer e is arrived at from the following relationships,

$$e * d = 1 \mod PHI(n)$$

'e' and 'x' must be found such that,

$$e * d + x * PHI(n) = gcd(d,PHI(n)) = 1 \qquad (4)$$

To help in finding the value of 'e' and 'x' in equation 4, use is made of an extension to Euclid's algorithm for calculating the gcd of two numbers, {24}. The algorithm follows.

```
   PROCEDURE GCD;

    u1 := 1;
    u2 := 0;
    u3 := d;
    v1 := 0;
    v2 := 1;
    v3 := PHI(n);
    DO WHILE v3 NOT = 0;
       q := u3/v3;
       t1 := u1 - q * v1;
       t2 := u2 - q * v2;
       t3 := u3 - q * v3;
       u1 := v1;
       u2 := v2;
       u3 := v3;
       v1 := t1;
       v2 := t2;
       v3 := t3;
    END;
    e := u2;
  END GCD;
```

EXAMPLE 6.2

The following is an example of using this method of message encryption.

The value of the message to be encrypted must be:

$$0 < M < (n-1)$$

A message was defined as equal to two characters of actual text. If each character of a message was replaced by its index, that is, A = 01, B = 02.......etc., the maximum value a message could have would be 2626. This would represent the two characters 'zz'. Consequently, n must not be smaller than 2626.

Let p = 53 and q = 59. Both are prime numbers.

$$\text{Then, } n = p * q$$
$$= 53 * 59 = 3127$$
$$\text{PHI}(n) = (p-1) * (q-1)$$
$$= 52 * 58 = 3016$$

Now, the value of d is picked such that 'e' is not less than $\log_2 n$. If the resulting value of 'e' does not meet this criteria, then another value for 'd' must be chosen until 'e' results in a value greater than $\log_2 n$.

For this example we choose d = 171.

The gcd(171,3016) = 1 as required in (2).

Now, to find e. From (4) we must find e and x such that,

$$171e + 3016x = \gcd(171,3016) = 1$$

Using algorithm GCD with d = 171 and PHI(n) = 3016 we find,

$$171 * 1411 + 3016 * (-80) = 1$$
$$241281 - 241280 = 1$$

Therefore, the encrypting key e = 1411, and the decrypting key d = 171. As a check, the requirement is that,

$$e > \log_2 3127$$
$$1411 > 6.64,$$

and that,

$e * d = 1 \bmod (p-1) * (q-1)$.

That is, $1411 * 171 = 1 \bmod 3016$

$241281 = 80 * 3016 + 1$.

Encrypting the text A L A G A R, we partition this transmission into messages of two characters each.

Clear text	A L	A G	A R
Numerically	01 12	01 07	01 18

Using the fast and efficient algorithm ENCDEC, described in the following pages, the clear text encodes to to the following. Using parameter values:

$n = 3127$

$e = 1411$

$M_1 = 112$, $M_2 = 107$, $M_3 = 118$

	A L	A G	A R
Cipher text	02 02	09 02	20 06

To decipher the above back to the original clear text, the same ENCDEC algorithm is used, except the parameter values used are: $n = 3127$

$d = 171$

$M_1 = 202$, $M_2 = 902$, $M_3 = 2006$

Clear text	01 12	01 07	01 18
	A L	A G	A R

Algorithm for Fast Encrypting or Decrypting.

This algorithm will compute the cipher or clear text using this method of encryption, or decryption. That is, it will calculate C or D in the following:

$$C = M^{e} \pmod n \qquad D = C^{d} \pmod n$$

At most:

- $2 * \log_2 e$ multiplications
- $2 * \log_2 e$ divisions

are required.

Program ENCDEC - Encode/Decode

```
PROCEDURE ENCDEC;
/* Let $e_k, e_{k-1}, e_{k-2}, e_{k-3}, \ldots\ldots, e_0$ be the binary bit */
/* configuration of the value e or d.                        */
    C := 1;
    FOR i = k to 0 by -1
    DO;
        C := C ** 2 (mod n);
        IF $e_i$ = 1
        THEN C := C * M (mod n)
    END;
        END ENCDEC;
```

The kay (e,n), in our example = (1411,3127), remained the same from message to message. The encoding function also remained the same. It was seen that in our example,

$$E_x(M_1) \neq E_x(M_2) \neq E_x(M_3)$$

6.12 Concluding Remarks

In this chapter, we have suggested another method for Public Key Crypto Systems whose security rests in part on the difficulty in factoring large numbers. Even so, the security of this system needs to be examined in more detail. In particular, the difficulty in factoring large numbers should be looked at very closely. If this method were to be implemented, exhaustive attempts to 'break' the system should be made. Once this method has withstood all attacks for a sufficient length of time, it may be used with a reasonable amount of confidence.

Comparing this method of encryption with others in preceeding chapters, we can generally say that it is just as difficult to unravel as any of the others. It is not only the method of encryption that makes the deciphering difficult, but also the values of the parameters used. This could include large prime numbers and complex permutations.

CHAPTER 7

CONCLUSION

The main intention in writing this thesis is to bring to light some of the problems the data processing industry has to cope with and investigate some sort of remedies to combat such problems. Here we speak of problems of a criminal nature. As was seen, the problems involved are serious and numerous. In this thesis we considered two areas of concern.

The first area of concern was the acquisition of software packages, application programs, and data without the rightful consent of the owners.

The second area of concern was the unauthorized penetration of highly secured telecommunication data networks for the sole purpose of siphoning off classified data.

We encrypted data according to various methods in in an attempt to show that there are ways of rendering data 'uninteresting' to the would-be thief. In making data networks as difficult as possible for unauthorized persons to access, it is hoped that such people would give up after spending a great deal of time attempting to gain entry into these data networks.

The passage of various laws to protect against the different data processing crimes, and the severe sentencing of those caught stealing, do not seem to be a significant deterent to first-time offenders or repeating offenders. This thievery should not be taken lightly. At one end of the scale, the existence of companies, whether large or small, may be in jeopardy while, at the other end of the scale, the freedom of entire countries may be at stake. Whether copying the logic of someone else's application program or breaking the secret telecommunications code, the loss to the owner can be insurmountable. Thus safeguards should be implemented as early as possible.

In discussing the protection and security of data and software, we analyzed various methods, namely, Self Encryption, Mathematical Text Manipulation, Superimposed Coding, and Substitution Permutation.

Of these four methods, it was difficult to single out one method that is superior to the others in all cases. In our study, we discovered that the way in which the data is used helps determine which method is most suitable. For example, the case of encrypting a decision table seems to lend itself best to the use of Superimposed Coding, while data to be transmitted is easily enciphered with the use of the Substitution Permutation or the Mathematical Text Manipulation method.

The degree of difficulty involved in unravel-ing the cipher text does not only lie in the method used. Depending on the imagination, cleverness, and intuition of the encryptor, the simplest of methods could become a complex enigma to the impostor. The method for text encryption that is associated with the factoring of a number into two large prime numbers depends on the selection of two very large, and supposedly difficult to find, prime numbers. The ability of an encrypted text to withstand discovery is influenced by two main factors. The first factor is the extent to which bits change positions as at the different levels of the Substitution Permutation Method. The greater this melange, the safer the encrypted text remains.

The second factor is the extent to which one avoids accidentally incorporating inherent cycles that could eventually assist the attacker.

Using Superimposed Coding, the problem of false drops arose. Two methods of reducing the number of false drops were suggested and analyzed. Our second method proved to be more successful. As stated in the text of this thesis, the better results were attributed mainly to the fact that a random selection was used in setting up the bit tables. In a more in depth study, it was seen that the frequency of false drops can be reduced without the loss of any encrypting power.

Turning now to the structure of telecommunication networks and the encryption of communicated data, we wanted to introduce this subject from its simplest form. A brief description of the major components of a teleprocessing network, led us to describe how a minimum cost network could be designed. The concept of the mix was introduced, and an analysis of the major role it plays in a network was presented in Chapter 5.

Networks having single and multiple authentication servers were discussed. It was found that the difference between the time required for a message to be processed in a network having multiple servers as opposed to having single servers, was not that large. Thus it was found that it was not worth while taking the extra risk of having only single server protection. The decision to go with multiple servers could very easily be reversed if costs can not be met. These variables along with others that enter into the design of a network, would have to be taken into consideration along with any peculiar characteristics of the intended network.

The portion of this thesis associated with the data networks was mainly involved in the encryption of the data that is to be sent by the originator to one or more receivers. In addition, the concept of digitalized signatures was analyzed, wherein it was emphasized that a receiver of a message must never have any doubts as to the authorship of the transmissions he receives. Chapter 6 presents one of probably many methods used in the verification procedure of the ownership of a message. The procedure outlined does not seem to have any major drawbacks, except that the many steps required to authenticate a {message,markings} set, would seem to take a great deal of time. The parameters that are required to be held in secrecy by both the sender and receiver, could very well make them vulnerable to attacks. In a large network where many take part and a large number of signed messages are transmitted, this method of authenticating signatures, as described in sections 6.4 to 6.10, may prove to be a bottleneck. Further refinement of this method could make it more acceptable.

A section on network accessing was introduced in order to bring to light the problems of trustworthiness among persons who must collaborate for successful accessing of networks or of classified files. Different methods were suggested, however, no one method met all the different criteria. Each case must be looked at and analyzed on an individual basis. The different cases have certain common denominators. Firstly, one should try to make access as difficult as possible so that the protecting part of the system can withstand the toughest attacks.

Secondly, one should arrange it so that the combinations of trustees are easily altered.

Thirdly, one should store as little data as possible that is required for the access analysis.

Lastly, a system should be set up whereby the pertinent personnel are notified as quickly as possible that an entry by 'force' is in progress.

Ideally, one should try to prevent one combination of trustees from collaborating in the use of a file or network for fraudulent purposes.

In the area of data encryption in a telecommunication network, we proposed certain methods. These were, Powers and Quotient Remainders, two encoding functions, and Substitution Permutation.

Here again, our goal was to scramble the data as much as possible so that an intruder could not unscramble the encrypted data. Each of the above methods has its own level of difficulty. We cannot say for certain that we have discovered the ultimate encoding function. This was not the intent of this thesis. Even with more sophisticated methods, the code is still broken. One has only to listen to recent news on how the United States had deciphered Canada's secret code. Certainly this area has a great deal of room for improvement.

A great deal of imagination is required to produce suitable 'protective walls' that will withstand continued attempts at unlawful accessing. Considerable imagination is also required in trying to make the transmission of data 'leakproof', thus preventing data from seeping out during transmission.

The main conclusion to be drawn from this research is that, as one prepares to install safeguards in different areas of data processing, one must remember that there will always be someone who will attempt to beat the system with great aspirations for penetrating the safeguards.

It is hoped that some of the methods described in this thesis will help deter unauthorized access to software packages and telecommunication networks.

REFERENCES

{ 1} BRANSTAD D.K., (1973)
"Security Aspects of Computer Networks",
AIAA Computer Network Systems Conference
Huntsville Alabama, April 16-18, 1973

{ 2} BRANDSTAD D.K., (1975)
"Encryption Protection in Computer Data Communications",
Proc. Fourth Data Communications Symp.,
October 1975, Pgs. 8.1-8.7

{ 3} CARLISLE J.H. (1976)
"Evaluating the Impact of Office Automation on Top Management Communication",
National Computer Conference
AFIPS 1976 Vol. 45, Pgs. 611-616

{ 4} CHAUM D.L. (1981)
"Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms",
CACM February 1981, Vol. 24, No. 2, Pgs. 84-88

{ 5} DIFFIE W., HELLMAN M.E. (1976)
"New Directions in Cryptography",
IEEE November 1976 Vol. IT-22, No. 6, Pgs. 644-654.

{ 6} HALL T.W. (1971)
"Implementation of an Interactive Conference System",
Spring Joint Computer Conference,
AFIPS 1971, Vol. 40, Pgs. 217-223

{ 7} HOLDEN J.B. (1980)
"Experiences of an Electronic Mail Vendor",
National Computer Conference 1980,
AFIPS, 1980, Vol. 49, Pgs. 493-497

{ 8} KLING R. (1978)
"Value Conflicts and Social Choice in Electronic Funds Transfer System Developments",
CACM August 1978, Vol. 21, No. 8, Pgs. 642-657

{ 9} LICKLIDER J.C.R., TAYLOR R.W. (1968)
"The Computer as a Communication Device",
Science & Technology April 1968, No. 76,
Pgs. 21-31

{10} LICKLIDER J.C.R., VEZZA A. (1978)
"Applications of Information Networks",
IEEE November 1976 Vol. 66, No. 11, Pgs. 1330-1345

{11} McQUILLAN J.M. (1978)
"Enhanced Message Addressing Capabilities for Computer Networks",
IEEE November 1978 Vol. 66, No. 11, Pgs. 1517-1527

{12} MERKLE R.C. (1978)
"Secure Communications Over Insecure Channels",
CACM April 1978 Vol. 21, No. 4, Pgs. 294-299.

{13} MORGAN H.L. (1976)
"Office Automation Project-A Research Perspective",
National Computer Conference 1980,
AFIPS, 1976, Vol. 45, Pgs. 605-610

{14} NEEDHAM R.M., SCHROEDER M.D. (1978)
"Using Encryption for Authentication in Large Networks of Computers",
CACM December 1978, Vol. 21, No. 12, Pgs. 993-999

{15} O'KELLEY H.E. (1980)
"Electronic Message System as a Function in the Integrated Electronic Office",
National Computer Conference 1980,
AFIPS, 1980, Vol. 49, Pgs. 499-502

{16} POPEK G.J., KLINE C.S., (1979)
"Encryption and Secure Computer Networks",
ACM Computing Surveys,
December 1979 Vol. 11, No. 4

{17} RIVEST R.I., SHAMIR A., ADLEMAN L. (1978)
"A Method for Obtaining Digital Signatures and Public-Key Cryptosystems",
CACM February 1978, Vol. 21, No. 2, Pgs. 120-126

{18} SHAMIR A. (1979)
"How to Share a Secret",
CACM November 1979 Vol. 22, No. 11, Pgs. 612-613.

{19} ULRICH W.E. (1980)
"Implementation Considerations in Electronic Mail",
National Computer Conference 1980,
AFIPS, 1980, Vol. 49, Pgs. 489-491

{20} ULRICH W.E. (1980)
"Introduction to Electronic Mail",
National Computer Conference 1980,
AFIPS, 1980, Vol. 49, Pgs. 485-488

BIBLIOGRAPHY

{21} DeMILLO R.A., LIPTON R. (1977)
"Proprietary Software Protection",
in Foundations of Secure Communication, Pgs. 115-129
R.J. DeMillo Ed., Academic Press, New York, 1978

{22} HOROWITZ E., SAHNI S.
"Fundamentals of Data Structures",
Computer Science Press Inc.

{23} KAM J.B., DAVIDA G.I. (1977)
"A Structured Design of Substitution-Permutation Encryption Network",
in Foundations of Secure Computation, Pgs. 95-113
R.J. DeMillo Ed., Academic Press, New York, 1978

{24} KNUTH D.E. (1969)
"The Art of Computer Programming" Vol. 2,
"Seminumerical Algorithms", Addison-Wesley,
Reading, Mass., 1969

{25} KNUTH D.E. (1973)
"The Art of Computer Programming" Vol. 3,
"Sorting and Searching", Pgs. 556-563,
Addison-Wesley, Reading, Mass., 1973

{26} POPEK G.J., KLINE C.S. (1977)
"Encryption Protocols, Public Key Algorithms, and Digital Signatures in Computer Networks",
in Fundations of Secure Communication, Pgs. 133-153
R.J. DeMillo Ed., Academic Press, New York, 1978

{27} RABIN R.O. (1977)
"DIGITALIZED SIGNATURES"
in Foundations of Secure Computation, Pgs. 155-168
R.J. DeMillo Ed., Academic Press, New York, 1978

{28} RIVEST R.L., ADLEMAN L., DERTOUZOS M.L. (1977)
"On Data Banks and Privacy Homomorphisms",
in Foundations of Secure Computation, Pgs. 169-177,
R.J. DeMillo Ed., Academic Press, New York, 1978

APPENDIX

PROGRAM LISTINGS

PROGRAM #1

```
C
C      PROGRAM TO OBTAIN A MINIMUM COST COMMUNICATION NET-
C     WORK.
C
      IMPLICIT INTEGER (A-Z)
      INTEGER*2 H(15)
      DIMENSION VRTEX1(100),VRTEX2(100),FREQ(100)
      COMMON R(100),RECSEQ(100),COST(100),COUNT(100),
     XVERTEX(100,100)
      PASS=0
      PASS2=0
C
C
C     OBTAIN TIME
C
  120 INDTME=0
      CALL CLOCK(INDTME,IUNITS,IDUM)
C
C     READ IN NUMBER OF EDGES
C
      READ (5,5,END=130) N
    5 FORMAT (I3)
C
C     INITIALIZATION
C
      DO 10 I=1,N
      R(I)=I
      COUNT(I)=1
   10 VERTEX(I,1)=I
      WRITE (6,45)
   45 FORMAT('-EDGES RETAINED    EDGES DISCARDED')
      TOTCOS=0
      COSSAV=0
C
C     READ EACH EDGE INTO INPUT ARRAYS
C
      READ (5,15) (VRTEX1(I),VRTEX2(I),COST(I),FREQ(I),
     XI=1,N)
   15 FORMAT (4(I3,1X))
C
C     THE SECOND PASS CALCULATES THE AVERAGE COST OF
C      COMMUNICATION USING FREQUENCY OF COMMUNICATION.
C
      IF (PASS) 90,90,95
   95 DO 85 I=1,N
   85 COST(I)=COST(I)*FREQ(I)
C
C     UPON RETURN FROM HSORT SUBROUTINE, RECSEQ
C     CONTAINS THE INDICES OF RECORDS IN NONDECREASING COST
C     (OR AVERAGE COST) SEQUENCE.
C
```

```
   90 CALL HSORT (N)
C
C     IN THE FOLLOWING, A SUBTREE WILL BE REFFERED TO
C     AS A SET.
C     SELECT THE NEXT PAIR OF VERTICES WHOSE EDGE HAS
C     THE NEXT MINIMUM COST.
C     COST.
C
      DO 20 INPUT=1,N
      ROOT1=VRTEX1(RECSEQ(INPUT))
C
C     LOCATE SET NO. WHERE VERTEX 1 EXISTS.

      CALL FIND (ROOT1)
C
      ROOT2=VRTEX2(RECSEQ(INPUT))
C
C     LOCATE SET NO. WHERE VERTEX 2 EXISTS.
C
      CALL FIND (ROOT2)
C
C     IF BOTH VERTEXES EXIST IN THE SAME SET, (IE. HAVE
C     THE SAME SET NO.), THEN DISCARD THE EDGE SINCE A CYCLE
C     WOULD BE CREATED OTHERWISE OBTAIN THE UNION OF BOTH
C     SETS.
C
      IF(ROOT1-ROOT2) 25,30,25
C
   25 WRITE (6,35) VRTEX1(RECSEQ(INPUT)),VRTEX2(RECSEQ
     X(INPUT))
   35 FORMAT (' ',1X,'(',I3,',',I3,')')
C
      CALL UNION (ROOT1,ROOT2)
C
      TOTCOS=TOTCOS+COST(RECSEQ(INPUT))
   20 CONTINUE
C
C     OBTAIN TIME
C
      INDTME = 999
      CALL CLOCK(INDTME,IUNITS,IDUM)
C
      WRITE (6,70) N,IUNITS
   70 FORMAT ('-','NUMBER OF EDGES = ',I3,5X,'CPU TIME UNITS
     XUNITS =',1X,I5)
      IF (PASS) 110,110,115
  110 WRITE (6,75) TOTCOS
   75 FORMAT('0TOTAL NETWORK COST $',I12)
      WRITE (6,80) COSSAV
   80 FORMAT('0NETWORK COST SAVED $',I12)
      GO TO 120
  115 WRITE (6,100) TOTCOS
  100 FORMAT('0TOTAL AVERAGE NETWORK COST $',I12)
      WRITE (6,105) COSSAV
```

```
 105 FORMAT('0NETWORK AVERAGE COST SAVED $',I12)
     GO TO 120
C
  30 WRITE (6,40) VRTEX1(RECSEQ(INPUT)),VRTEX2(RECSEQ
    X(INPUT))
  40 FORMAT (' ',19X,'(',I3,',',I3,')')
     COSSAV=COSSAV+COST(RECSEQ(INPUT))
     GO TO 20
 130 IF(PASS.EQ.1) GO TO 135
     REWIND 5
     PASS=1
     GO TO 120
 135 IF (PASS2.EQ.1) STOP
     PASS=0
     REWIND 5
     PASS2=1
     GO TO 120
     END
     SUBROUTINE HSORT (N)
C
C    HEAP SORT SUBROUTINE
C
     IMPLICIT INTEGER (A-Z)
     COMMON R(100),RECSEQ(100),COST(100),COUNT(100),
    XVERTEX(100,100)
     I=N/2
  10 IF (I) 30,30,20
  20 CALL ADJUST (I,N)
     I=I-1
     GO TO 10
  30 RECSEQ(N)=R(1)
     I=N-1
  60 IF (I) 50,50,40
  40 T=R(I+1)
     R(I+1)=R(1)
     R(1)=T
C
     CALL ADJUST(1,I)
C
     RECSEQ(I)=R(1)
     I=I-1
     GO TO 60
  50 RETURN
     END
     SUBROUTINE ADJUST (I,N)
     IMPLICIT INTEGER (A-Z)
     COMMON R(100),RECSEQ(100),COST(100),COUNT(100),
    XVERTEX(100,100)
C
     RR=R(I)
     COSTI=COST(R(I))
     J=2*I
  10 IF(J-N) 1,1,2
   1 IF(J-N) 3,5,5
```

```
    3 IF(COST(R(J))-COST(R(J+1))) 4,5,5
    4 J=J+1
    5 IF(COSTI-COST(R(J))) 7,6,6
    6 R(J/2)=RR
      RETURN
    7 R(J/2)=R(J)
      J=2*J
      GO TO 10
    2 R(J/2)=RR
      RETURN
      END
      SUBROUTINE FIND (ROOT)
C
C
      IMPLICIT INTEGER (A-Z)
      COMMON R(100),RECSEQ(100),COST(100),COUNT(100),
     XVERTEX(100,100)
C
      DO 5 I=1,100
      IF(COUNT(I)) 5,5,20
   20 K=COUNT(I)
      DO 10 J=1,K
      IF (ROOT-VERTEX(I,J)) 10,30,10
   10 CONTINUE
    5 CONTINUE
      STOP 1
   30 ROOT=I
      URN

      ROUTINE UNION (ROOT1,ROOT2)
C
C
      IMPLICIT INTEGER (A-Z)
      COMMON R(100),RECSEQ(100),COST(100),COUNT(100),
     XVERTEX(100,100)
C
      IF (COUNT(ROOT1)-COUNT(ROOT2)) 5,10,10
    5 K=COUNT(ROOT1)
      DO 15 J=1,K
      VERTEX(ROOT2,(COUNT(ROOT2)+1))=VERTEX(ROOT1,J)
   15 COUNT(ROOT2)=COUNT(ROOT2)+1
      COUNT(ROOT1)=0
      RETURN
   10 K=COUNT(ROOT2)
      DO 20 J=1,K
      VERTEX(ROOT1,(COUNT(ROOT1)+1))=VERTEX(ROOT2,J)
   20 COUNT(ROOT1)=COUNT(ROOT1)+1
      COUNT(ROOT2)=0
      RETURN
      END
```

PROGRAM #2

```
C
C      PROGRAM TO DETERMINE THE RELATIONSHIP BETWEEN
C      L(M) AND PROCESSING TIME FOR DIFFERENT S, T,
C      AND R PARAMETER VALUES USED IN DIGITAL SIG-
C      NATURE AUTHENTICATION.
C
       INTEGER SIGN,RNUMS,S,T,R,ENCOD1,ENCOD2,ALPHA
       COMMON LETTER(82000),MSGVAL(82000),KEYS(200,2),
      XSIGN(70),RNUMS(50)
       COMMON ALPHA(200)
       DIMENSION LARAY(175),ISARAY(175),ITARAY(175),IRARAY(175)
C
C      READ PARAMETERS L(M),S,T,R INTO ARRAYS
C
       DO 60 I=1,174
60     READ (5,10)  LARAY(I),ISARAY(I),ITARAY(I),IRARAY(I)
10     FORMAT (I5,1X,I3,1X,I2,1X,I2)
C
C      READ IN MESSAGE CHARACTER BY CHARACTER
C
       MSGEND=LARAY(I)
       READ (5,20) (LETTER(IL),IL=1,81920)
20     FORMAT (80A1)
       WRITE(6,62)
62     FORMAT(' L(M)',5X,'S',4X,'T',3X,'R',1X,'TIME IN CPU UNITS')
       DO 40 I=1,174
       INDIC=0
       CALL CLOCK(INDIC,IUNITS,IDUM)
C
C      TRANSLATE MESSAGE TO NUMERIC VALUES
C
       CALL TRANS (LARAY(I))
C
C      COMPRESS THE MESSAGE, C(M)
C
       CALL COMPRS (LARAY(I),ENCOD1,ENCOD2)
C
C      GENERATE THE MARKINGS THAT MAKE UP THE SIGNATURE,
C        AND CALCULATE THE ALPHA I'S.
C
       CALL SIGTRE(ISARAY(I),ITARAY(I),ENCOD1,ENCOD2)
C
C      VERIFY RANDOM KEYS SELECTED
C
       CALL VERFY1 (ITARAY(I),IRARAY(I))
C
C      VERIFICATION OF SIGNATURE
C
       CALL VERFY2 (IRARAY(I),ENCOD1,ENCOD2)
C
C
```

```
      INDIC=999
      CALL CLOCK(INDIC,IUNITS,IDUM)
40    WRITE(6,36) LARAY(I),ISARAY(I),ITARAY(I),IRARAY(I),IUNITS
36    FORMAT(' ',I5,3X,I3,2X,I2,2X,I2,2X,I6)
      STOP
      END
      SUBROUTINE TRANS (MSGLTH)
      COMMON LETTER(82000),MSGVAL(82000),KEYS(200,2),
     XSIGN(70),RNUMS(50)
      COMMON ALPHA(200)
      DATA IBLANK,IA,IB,IC,ID,IE,IF,IG,IH,II,IJ,IK,IL,IM,
     XIN,IO,IP,IQ,
     1IR,IS,IT,IU,IV,IW,IX,IY,IZ/' ','A','B','C','D','E','F',
     2'G','H','I','J','K','L','M','N','O','P','Q','R','S',
     3'T','U','V','W','X','Y','Z'/
C
C
      DO 1 I=1,MSGLTH
      IF (LETTER(I).EQ.IBLANK) MSGVAL(I)=0
      IF (LETTER(I).EQ.IA) MSGVAL(I)=1
      IF (LETTER(I).EQ.IB) MSGVAL(I)=2
      IF (LETTER(I).EQ.IC) MSGVAL(I)=3
      IF (LETTER(I).EQ.ID) MSGVAL(I)=4
      IF (LETTER(I).EQ.IE) MSGVAL(I)=5
      IF (LETTER(I).EQ.IF) MSGVAL(I)=6
      IF (LETTER(I).EQ.IG) MSGVAL(I)=7
      IF (LETTER(I).EQ.IH) MSGVAL(I)=8
      IF (LETTER(I).EQ.II) MSGVAL(I)=9
      IF (LETTER(I).EQ.IJ) MSGVAL(I)=10
      IF (LETTER(I).EQ.IK) MSGVAL(I)=11
      IF (LETTER(I).EQ.IL) MSGVAL(I)=12
      IF (LETTER(I).EQ.IM) MSGVAL(I)=13
      IF (LETTER(I).EQ.IN) MSGVAL(I)=14
      IF (LETTER(I).EQ.IO) MSGVAL(I)=15
      IF (LETTER(I).EQ.IP) MSGVAL(I)=16
      IF (LETTER(I).EQ.IQ) MSGVAL(I)=17
      IF (LETTER(I).EQ.IR) MSGVAL(I)=18
      IF (LETTER(I).EQ.IS) MSGVAL(I)=19
      IF (LETTER(I).EQ.IT) MSGVAL(I)=20
      IF (LETTER(I).EQ.IU) MSGVAL(I)=21
      IF (LETTER(I).EQ.IV) MSGVAL(I)=22
      IF (LETTER(I).EQ.IW) MSGVAL(I)=23
      IF (LETTER(I).EQ.IX) MSGVAL(I)=24
      IF (LETTER(I).EQ.IY) MSGVAL(I)=25
      IF (LETTER(I).EQ.IZ) MSGVAL(I)=26
      CONTINUE
      RETURN
      END
      SUBROUTINE COMPRS(M,E1,E2)
      INTEGER E1,E2,WM,ALPHA,SIGN,RNUMS
      COMMON LETTER(82000),MSGVAL(82000),KEYS(200,2),
     XSIGN(70),RNUMS(50)
      COMMON ALPHA(200)
C
```

```
      I=M
      IC3=56
      IC4=32
1     IC1=MSGVAL(I-1)
      IC2=MSGVAL(I)
      CALL FUNCTN(IC1,IC2,IC3,IC4)
      I=I-2
      IF(I.GT.0) GO TO 1
C     ICOMP=(100*IC3)+IC4
C     WRITE(6,3) ICOMP
C     FORMAT('-VALUE OF COMPRESSED MESSAGE = ',I5)
      RETURN
      END
      SUBROUTINE SIGTRE (S,T,ENCOD1,ENCOD2)
      INTEGER S,ENCOD1,ENCOD2,SIGN,RNUMS,EN1,EN2,CMOI,
     XCMOIP1,CMOIP2
      INTEGER ALP1,ALP2,T,CM,ALPHA
      COMMON LETTER(82000),MSGVAL(82000),KEYS(200,2),
     XSIGN(70),RNUMS(50)
      COMMON ALPHA(200)
C
C     GENERATE 'S' KEYS
C
      L=53
      DO 1 I=1,S
      DO 2 J=1,2
3     F = RANDOM(L)
      F = F * 100.0
      N = F
      IF (N.GT.26) GO TO 3
      IF (N.LE.0) GO TO 3
2     KEYS(I,J)=N
1     CONTINUE
C
C     CHECK FOR DUPLICATE KEYS
C
      IEND=S-1
8     DO 10 I=1,IEND
      IX=I+1
      DO 11 J=IX,S
      IF(KEYS(I,1).NE.KEYS(J,1)) GO TO 11
      IF(KEYS(I,2).NE.KEYS(J,2)) GO TO 11
7     F=RANDOM(L)
      F=F*100.0
      N=F
      IF(N.GT.26) GO TO 7
      IF(N.LE.0) GO TO 7
      KEYS(J,1)=N
      GO TO 8
11    CONTINUE
10    CONTINUE
C
C     COMPUTE THE ALPHA LIST OF S ALPHA I'S USING
C     KEYS & C(MO(I))'S
```

```
C
      DO 5 I=1,S
      CMOI=5632 + I
      CMOIP1=CMOI/100
      CMOIP2=(CMOI-(CMOIP1*100))
     CALL FUNCTN (KEYS(I,1),KEYS(I,2),CMOIP1,CMOIP2)
5     ALPHA(I)=(CMOIP1*100)+CMOIP2
C
C     GENERATE THE SIGNATURE'S MARKINGS
C
      DO 4 I=1,T
      EN1=ENCOD1
      EN2=ENCOD2
      CALL FUNCTN (KEYS(I,1),KEYS(I,2),EN1,EN2)
4     SIGN(I)=(100*EN1)+EN2
      RETURN
      END
      SUBROUTINE VERFY1(T,R)
      COMMON LETTER(82000),MSGVAL(82000),KEYS(200,2),
     XSIGN(70),RNUMS(50)
      COMMON ALPHA(200)
      INTEGER T,R,SIGN,RNUMS,ALPHA,CMOI,CMOIP1,
     XCMOIP2,ALP1,ALP2
      L=13
C
C     GENERATE R RANDOM NUMBERS
C
      DO 4 I=1,R
3     F=RANDOM(L)
      F = F * 100.0
      N = F
      IF(N.GT.T) GO TO 3
      IF(N.LE.0) GO TO 3
4     RNUMS(I)=N
C
C     CHECK FOR DUPLICATE RANDOM NUMBERS
C
      IEND=R-1
5     DO 6 I=1,IEND
      IX=I+1
      DO 7 J=IX,R
      IF(RNUMS(I).NE.RNUMS(J)) GO TO 7
8     F=RANDOM(L)
      F=F*100.0
      N=F
      IF(N.GT.T) GO TO 8
      IF(N.LE.0) GO TO 8
      RNUMS(J)=N
      GO TO 5
7     CONTINUE
6     CONTINUE
      MATCH=0
      DO 9 I=1,R
      CMOI=5632+RNUMS(I)
      CMOIP1=CMOI/100
      CMOIP2=(CMOI-(CMOIP1*100))
      CALL FUNCTN (KEYS(RNUMS(I),1),KEYS(RNUMS(I),2),
     XCMOIP1,CMOIP2)
      ITEST=(CMOIP1*100)+CMOIP2
      IF(ITEST.EQ.ALPHA(RNUMS(I))) GO TO 9
      MATCH=1
9     CONTINUE
      IF (MATCH.EQ.0) RETURN
      WRITE(6,11)
11    FORMAT('0 LEVEL 1 KEY VERIFICATION FAILED')
      RETURN
      END
      SUBROUTINE VERFY2(R,ENCOD1,ENCOD2)
      INTEGER R,ENCOD1,ENCOD2,EN1,EN2,RNUMS,KEYS,
     XSIGN,ALPHA,CM
      COMMON LETTER(82000),MSGVAL(82000),KEYS(200,2),
     XSIGN(70),RNUMS(50)
      COMMON ALPHA(200)
C
      MATCH=0
      DO 2 I=1,R
      EN1=ENCOD1
      EN2=ENCOD2
      CALL FUNCTN (KEYS(RNUMS(I),1),KEYS(RNUMS(I),2),
     XEN1,EN2)
      ITEST=(100*EN1)+EN2
      IF(ITEST.EQ.SIGN(RNUMS(I))) GO TO 2
      MATCH=1
2     CONTINUE
      IF(MATCH.EQ.0) RETURN
      WRITE (6,6)
6     FORMAT('0LEVEL 2 VERIFICATION FAILED')
      RETURN
      END
      FUNCTION RANDOM(ISEED)
      INTEGER ICON/Z7FFFFFFF/
      IF(ISEED.EQ.0) ISEED=39
      ISEED=ISEED*23*13**4
      IF(ISEED.LT.0) ISEED=ISEED+ICON+1
      RANDOM=ISEED/FLOAT(ICON)
      RETURN
      END
      SUBROUTINE FUNCTN (IC1,IC2,IC3,IC4)
      DOUBLE PRECISION X,Y,Z
      IF(IC1.EQ.IC3) GO TO 1
      X=IABS(IC1-IC3)
      Y=X**IC1
5     Z=DMOD(Y,29.0D0)
      IC3=Z
      IF(IC2.EQ.IC4) GO TO 2
      X=IABS(IC2-IC4)
      Y=X**IC2
7     Z=DMOD(X,29.0D0)
```

```
      IC4=Z
      RETURN
1     IF(IC1.EQ.IC2) GO TO 3
      X=IABS(IC1-IC2)
      Y=X**IC3
      GO TO 5
2     IF(IC3.EQ.IC4) GO TO 4
      X=IABS(IC3-IC4)
      Y=X**IC1
      GO TO 7
3     IF(IC1.EQ.IC4) GO TO 6
      X=IABS(IC1-IC4)
      Y=X**IC3
      GO TO 5
4     X=IC1
      Y=X**IC1
      GO TO 7
6     X=IC1
      Y=X**IC1
      GO TO 5
      END
```

PROGRAM #3

```
C
C     PROGRAM TO SIMULATE THE CONVENTIONAL AND PUBLIC KEY
C     PROTOCOL SYSTEMS IN A MULTI-AUTHENTICATION SERVER
C     DATA NETWORK.
C
      INTEGER TS1,TS2,DESTIN,TEK,PARM1,PARM2,PARM3,
     XPARM4,PARM5,PARM6,
     1        ATHSER,ASKEY,SECKEY,ORIGIN,PUBKEY
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
      DEFINE FILE 12(1,80,E,IASSVR)
      INTEGER*2 H(15)
C
C     INITIALIZE THE 4 TABLES:
C               - AUTHENTICATION SERVERS
C               - ATHENTICATION SERVERS' SECRET KEYS
C               - PARTICIPANTS' SECRET KEYS
C               - PARTICIPANTS' PUBLIC KEYS
C
      READ(5,5) (ASKEY(I),I=1,4)
    5 FORMAT(4I2)
      READ(5,5) (SECKEY(I),I=1,4)
      READ(5,5) (PUBKEY(I),I=1,4)
      READ (5,10) (((ATHSER(I,J,K),K=1,5),J=1,4),I=1,4)
   10 FORMAT(80I1)
   64 ID=0
   65 INDIC=0
      CALL CLOCK(INDIC,IUNITS,IDUM)
      READ(5,15,END=200) ORIGIN,DESTIN
   15 FORMAT(2I1)
      IF(ID.EQ.0) WRITE (6,75)
   75 FORMAT('-   EXAMPLE OF CONVENTIONAL ENCRYPTED
     XAUTHENTICATION')
      IF(ID.EQ.1) WRITE (6,80)
   80 FORMAT('-   EXAMPLE OF PUBLIC KEY ENCRYPTED
     XAUTHENTICATION')
      WRITE(6,20) ORIGIN,DESTIN
   20 FORMAT('-',4X,'ORIGIN NODE = ',I1,3X,
     X'DESTINATION NODE = ',I1)
      CALL TIMSTP(TS1)
C
C     A --> ASA
C
C
C     COMMUNICATION BETWEEN AUTHENICATION SERVERS INVOLVED BETWEEN
C     ORIGINATOR AND RECEIVER
C
C     ASA <--> ASB <--> ASC <--> ASD <--> .......
C
      CALL TXTKEY(TEK)
      IF(ID.EQ.0) WRITE(6,21)ORIGIN,ORIGIN,ORIGIN,DESTIN,TS1
   21 FORMAT('-',7X,I1,' --> AS',I1,3X,'{',I1,',',I1,',',
  XI4,'}')
   IF(ID.EQ.1) WRITE(6,22) ORIGIN,ORIGIN,ORIGIN,DESTIN
22 FORMAT('-',7X,I1,' --> AS',I1,3X,'{',I1,',',I1,'}')
   I=1
   PARM1=ORIGIN
   PARM2=DESTIN
   PARM3=TEK
   PARM4=TS1
25 CALL ENCR1(PARM1,PARM2,PARM3,PARM4,I,ORIGIN,DESTIN,ID)
   J=I+1
   IF(ID.EQ.0) WRITE(6,30) I,J,PARM1,PARM2,PARM3,PARM4
30 FORMAT('-',5X,'AS',I1,' --(ENCRYPTED)-->  AS',I1,
  X3X,'{',I4,
  X3(',',I4),'}')
   IF(ID.EQ.1) WRITE(6,31) I,J,PARM1,PARM2,PARM3
31 FORMAT('-',5X,'AS',I1,' --(ENCRYPTED)-->  AS',I1,
  X3X,'{',I4,
  X2(',',I4),'}')
   CALL DECR1(PARM1,PARM2,PARM3,PARM4,I,ORIGIN,
  XDESTIN,ID)
   I=I+1
   IX=I+1
   IF(ATHSER(ORIGIN,DESTIN,IX).NE.0) GO TO 25
   K=I
   PARM5=TEK
   PARM6=ORIGIN
35 CALL ENCR2(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6,
  XORIGIN,DESTIN,I,ID)
   J=I-1
   IF(ID.EQ.0) WRITE(6,40) I,J,PARM5,PARM6,PARM1,
  XPARM2,PARM3,PARM4
40 FORMAT('-',5X,'AS',I1,' --(ENCRYPTED)-->  AS',I1,
  X3X,'{(',I4,',',
  XI4,')',4(',',I4),'}')
   IF(ID.EQ.1) WRITE(6,41) I,J,PARM1,PARM2,PARM3
41 FORMAT('-',5X,'AS',I1,' --(ENCRYPTED)-->  AS',I1,
  X3X,'{',I4,
  X2(',',I4),'}')
   CALL DECR2(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6,
     XORIGIN,DESTIN,I,ID)
   I=I-1
   IF(I.GT.1) GO TO 35
C
C     ASA --> A
C
   CALL ENCR3(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6,
  XORIGIN,DESTIN,ID)
   IF(ID.EQ.0) WRITE(6,45) ATHSER(ORIGIN,DESTIN,1),
  XORIGIN,PARM5,
  XPARM6,PARM1,PARM2,PARM3,PARM4
45 FORMAT('-',5X,'AS',I1,' --(ENCRYPTED)-->  ',I1,
  X3X,'{(',I4,',',
  XI4,')',4(',',I4),'}')
   IF(ID.EQ.1) WRITE(6,46) ATHSER(ORIGIN,DESTIN,1),
     XORIGIN,PARM1,
     XPARM2,PARM3
  46 FORMAT('-',5X,'AS',I1,' --(ENCRYPTED)-->  ',I1,
     X3X,'{',I4,
     X2(',',I4),'}')
      CALL DECR3(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6,
     XORIGIN,DESTIN,ID)
C
C     A --> B
C
      IF(ID.EQ.1) GO TO 85
      CALL RCVER(PARM5,PARM6,TS2,DESTIN)
      WRITE(6,50) ORIGIN,DESTIN,PARM5,PARM6,TS2
  50  FORMAT('-',7X,I1,'  --(ENCRYPTED)-->  ',I1,
     X3X,'{',I4,',',I4,'}',
     X',','{',I6,'}')
C
C     B --> A
C
      CALL SENDER(PARM3,TS2)
      WRITE(6,55) DESTIN,ORIGIN,TS2
  55  FORMAT('-',7X,I1,'  --(ENCRYPTED)-->  ',I1,
     X3X,'{',I7,'}')
      WRITE(6,60)
  60  FORMAT('-END OF TRANSMISSION')
      ID=1
      INDIC=999
      CALL CLOCK(INDIC,IUNITS,IDUM)
      WRITE(6,135) IUNITS
 135  FORMAT('-  PROTOCOL ENCRYPTION TIME -
     XCONVENTIONAL ',I4)
      GO TO 65
  85  CALL TIMSTP(TS1)
      CALL ENCR4(PARM1,PARM4,DESTIN)
      WRITE(6,90) ORIGIN,DESTIN,PARM1,PARM4
  90  FORMAT('-',7X,I1,'  --(ENCRYPTED)-->  ',I1,
     X3X,'{',I4,',',I4,'}')
      CALL DECR4(PARM1,PARM4,DESTIN)
      WRITE(6,95) DESTIN,ATHSER(ORIGIN,DESTIN,K),
     XORIGIN,DESTIN
  95  FORMAT('-',7X,I1,'  --(CLEAR TEXT)-->  AS',I1,
     X3X,'{',I1,',',I1,
     X     '}')
      I=K
 100  J=I-1
      CALL ENCR2(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6,I,ID)
      WRITE(6,41) ATHSER(ORIGIN,DESTIN,I),
     XATHSER(ORIGIN,DESTIN,J),
     X          PARM1,PARM2,PARM3
      CALL DECR2(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6,I,ID)
      I=I-1
      IF(I.GT.1) GO TO 100
 110  IX=I+1
      IF(ATHSER(ORIGIN,DESTIN,IX).EQ.0) GO TO 120
      J=I+1
      CALL ENCR1(PARM1,PARM2,PARM3,PARM4,I,ORIGIN,DESTIN,ID)
      WRITE(6,41) ATHSER(ORIGIN,DESTIN,I),
     XATHSER(ORIGIN,DESTIN,J),
     X            PARM1,PARM2,PARM3
      CALL DECR1(PARM1,PARM2,PARM3,PARM4,I,ORIGIN,DESTIN,ID)
      I=I+1
      GO TO 110
  120 CALL ENCR1(PARM1,PARM2,PARM3,PARM4,I,ORIGIN,DESTIN,ID)
      WRITE(6,46) ATHSER(ORIGIN,DESTIN,I),DESTIN,PARM1,
     XPARM2,PARM3
      CALL DECR1(PARM1,PARM2,PARM3,PARM4,I,ORIGIN,DESTIN,ID)
      CALL TIMSTP(TS2)
      PARM4=TS2
      CALL ENCR4(PARM2,PARM4,ORIGIN)
      WRITE(6,130) DESTIN,ORIGIN,PARM2,PARM4
  130 FORMAT('-',7X,I1,'  --(ENCRYPTED)-->  ',I1,
     X3X,'{',I1,',',I6,'}')
      WRITE(6,60)
      INDIC=999
      CALL CLOCK(INDIC,IUNITS,IDUM)
      WRITE(6,140) IUNITS
  140 FORMAT('-  PROTOCOL ENCRYPTION TIME - PUBLIC KEY ',I4)
      GO TO 64
  200 STOP
      END
      SUBROUTINE TIMSTP(ISTAMP)
      INTEGER*2 H(15)
      CALL TIME(H)
      WRITE(12'1,5) H
    5 FORMAT(15A2)
      READ(12'1,10) IHR,IMIN,ISEC,IHSEC
   10 FORMAT(4I2)
      ISTAMP=(IHR*100)+IMIN
      RETURN
      END
      SUBROUTINE TXTKEY(TEK)
      INTEGER TEK
      L=13
C
C     GENERATE THE TEK
C
    5 F=RANDOM(L)
      F=F*100.0
      TEK=F
      IF(TEK.LE.0) GO TO 5
      IF(TEK.GT.25) GO TO 5
      RETURN
      END
      SUBROUTINE ENCR1(PARM1,PARM2,PARM3,PARM4,I,
     XORIGIN,DESTIN,ID)
      IMPLICIT INTEGER(A-Z)
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
C
```

```
C
      IF(ID.EQ.1) GO TO 5
      PARM1=PARM1*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM2=PARM2*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM3=PARM3*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM4=PARM4*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      RETURN
    5 PARM1=PARM1*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM2=PARM2*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM3=PUBKEY(ORIGIN)*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      RETURN
      END
      SUBROUTINE DECR1(PARM1,PARM2,PARM3,PARM4,I,
     XORIGIN,DESTIN,ID)
      IMPLICIT INTEGER(A-Z)
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
C
C
      IF(ID.EQ.1) GO TO 5
      PARM1=PARM1/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM2=PARM2/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM3=PARM3/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM4=PARM4/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      RETURN
    5 PARM1=PARM1/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM2=PARM2/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM3=PARM3/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      RETURN
      END
      SUBROUTINE ENCR2(PARM1,PARM2,PARM3,PARM4,PARM5,
     XPARM6,ORIGIN,
     XDESTIN,I,ID)
      IMPLICIT INTEGER(A-Z)
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
C
C
      IF(ID.EQ.1) GO TO 5
      CALL ENCR1(PARM1,PARM2,PARM3,PARM4,I,ORIGIN,DESTIN,ID)
      PARM5=PARM5*SECKEY(I)
      PARM6=PARM6*SECKEY(I)
      PARM5=PARM5*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM6=PARM6*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      RETURN
    5 PARM1=PARM1*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM2=PARM2*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM3=PUBKEY(DESTIN)*ASKEY(ATHSER(ORIGIN,DESTIN,I))
      RETURN
      END
      SUBROUTINE DECR2(PARM1,PARM2,PARM3,PARM4,PARM5,
     XPARM6,ORIGIN,
     XDESTIN,I,ID)
      IMPLICIT INTEGER(A-Z)
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
C
```

```
C
      IF(ID.EQ.1) GO TO 5
      CALL DECR1(PARM1,PARM2,PARM3,PARM4,I,ORIGIN,DESTIN,ID)
      PARM5=PARM5/SECKEY(I)
      PARM6=PARM6/SECKEY(I)
      PARM5=PARM5/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM6=PARM6/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      RETURN
    5 PARM1=PARM1/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM2=PARM2/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      PARM3=PARM3/ASKEY(ATHSER(ORIGIN,DESTIN,I))
      RETURN
      END
      SUBROUTINE ENCR3(PARM1,PARM2,PARM3,PARM4,PARM5,
     XPARM6,ORIGIN,
     XDESTIN,ID)
      IMPLICIT INTEGER(A-Z)
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
C
C
      IF(ID.EQ.1) GO TO 5
      PARM5=PARM5*SECKEY(DESTIN)
      PARM6=PARM6*SECKEY(DESTIN)
      PARM5=PARM5*SECKEY(ORIGIN)
      PARM6=PARM6*SECKEY(ORIGIN)
      PARM1=PARM1*SECKEY(ORIGIN)
      PARM2=PARM2*SECKEY(ORIGIN)
      PARM3=PARM3*SECKEY(ORIGIN)
      PARM4=PARM4*SECKEY(ORIGIN)
      RETURN
    5 PARM1=PARM1*ASKEY(ATHSER(ORIGIN,DESTIN,1))
      PARM2=PARM2*ASKEY(ATHSER(ORIGIN,DESTIN,1))
      PARM3=PUBKEY(DESTIN)*ASKEY(ATHSER(ORIGIN,DESTIN,1))
      RETURN
      END
      SUBROUTINE DECR3(PARM1,PARM2,PARM3,PARM4,PARM5,
     XPARM6,ORIGIN,
     XDESTIN,ID)
      IMPLICIT INTEGER(A-Z)
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
C
C
      IF(ID.EQ.1) GO TO 5
      PARM5=PARM5/SECKEY(DESTIN)
      PARM6=PARM6/SECKEY(DESTIN)
      PARM5=PARM5/SECKEY(ORIGIN)
      PARM6=PARM6/SECKEY(ORIGIN)
      PARM1=PARM1/SECKEY(ORIGIN)
      PARM2=PARM2/SECKEY(ORIGIN)
      PARM3=PARM3/SECKEY(ORIGIN)
      PARM4=PARM4/SECKEY(ORIGIN)
      RETURN
    5 PARM1=PARM1/ASKEY(ATHSER(ORIGIN,DESTIN,1))
      PARM2=PARM2/ASKEY(ATHSER(ORIGIN,DESTIN,1))
      PARM3=PARM3/ASKEY(ATHSER(ORIGIN,DESTIN,1))
      RETURN
      END
      SUBROUTINE ENCR4(PARMX,PARM4,PARMY)
      IMPLICIT INTEGER(A-Z)
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
C
C
      PARMX=PARMX*PUBKEY(PARMY)
      PARM4=PARM4*PUBKEY(PARMY)
      RETURN
      END
      SUBROUTINE DECR4(PARMX,PARM4,PARMY)
      IMPLICIT INTEGER(A-Z)
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
C
C
      PARMX=PARMX/PUBKEY(PARMY)
      PARM4=PARM4/PUBKEY(PARMY)
      RETURN
      END
      SUBROUTINE RCVER(PARM5,PARM6,TS2,DESTIN)
      IMPLICIT INTEGER(A-Z)
      COMMON ATHSER(4,4,5),ASKEY(4),SECKEY(4),PUBKEY(4)
      CALL TIMSTP(TS2)
      TS2=TS2*PARM5
      PARM5=PARM5*SECKEY(DESTIN)
      PARM6=PARM6*SECKEY(DESTIN)
      RETURN
      END
      SUBROUTINE SENDER(PARM3,TS2)
      IMPLICIT INTEGER(A-Z)
      TS2=TS2*PARM3
      RETURN
      END
      FUNCTION RANDOM(ISEED)
      INTEGER ICON/Z7FFFFFFF/
      IF(ISEED.EQ.0) ISEED=39
      ISEED=ISEED*23*13**4
      IF(ISEED.LT.0) ISEED=ISEED+ICON+1
      RANDOM=ISEED/FLOAT(ICON)
      RETURN
      END
```

PROGRAM #4

```
*
*  PROGRAM TO COMPUTE THE EXECUTION TIME ELAPSED
*
CLOCK    CSECT
         SAVE  (14,12)
         BALR  12,0
         USING *,12
         ST    1,PARMADR     SAVE ADDRESS OF PARAMETER LIST
         ST    13,SAVE+4
         LA    4,SAVE
         ST    4,8(13)
         L     5,4(1)
         ST    5,ARGADR      SAVE PARAMETER ADDRESS
         L     5,0(1)        LOAD ADDRESS OF FIRST PARAMETER
         L     5,0(5)        LOAD FIRST PARAMETER = FIRST
*                                                   TIME IND
         LTR   5,5           IS IT THE FIRST CALL (1ST.=0)
         BNZ   SECOND        NO
         STIMER TASK,TUINTVL=CLOCKTIM
         B     LEAVE         RETURN TO CALLING PROGRAM
SECOND   TTIMER CANCEL       REG. 0 CONTAINS REMAINING TIME
         L     7,CLOCKTIM    REG. 7 = INITIAL CLOCK TIME
         SR    7,0           REG. 7 = TIME USED
         L     4,ARGADR
         ST    7,0(4)
         L     1,PARMADR
LEAVE    L     13,SAVE+4
         L     14,12(13)
         XR    15,15         NO RETURN CODE
         LM    2,12,28(13)
         BR    14            RETURN TO CALLING PROGRAM
*
         DS    0F
SAVE     DS    18F
CLOCKTIM DC    F'262143'
ARGADR   DC    A(0)
PARMADR  DC    A(0)             DUMMY TO END PARM LIST
         DROP  12
         END   CLOCK
```

PROGRAM #5

```
C
C     PROGRAM TO DETERMINE THE NUMBER OF FALSE DROPS AS
C     AS A RESULT OF OPERATING ON A DECISION TABLE.
C
      INTEGER PAIRS,COMBIN,POSWDS,POSTOT,WRDTOT,WRDSZE
      DIMENSION PAIRS(32768,4),POSWDS(15,20),NEGWDS(15,20)
      COMMON COMBIN(15)
C
C
    1 READ (5,2,END=180) N,WRDSZE,MAXBIT,MINBIT
    2 FORMAT (4(I2,1X))
      WRITE(6,6) N,WRDSZE,MAXBIT,MINBIT
    6 FORMAT('1','NUMBER OF WORDS = ',I2,2X,'WORD SIZE = ',
     XI2,
     1' BITS  MAXIMUM BITS ON = ',I2,2X,'MINIMUM BITS ON = '
     X,I2)
      DO 3 I=1,N
      DO 4 J=1,WRDSZE
      POSWDS(I,J)=0
    4 NEGWDS(I,J)=0
    3 CONTINUE
C
C    SET BITS ON RANDOMLY
C
      ISEED=73
      CALL BITSET(POSWDS,N,WRDSZE,MAXBIT,MINBIT,ISEED)
      CALL BITSET(NEGWDS,N,WRDSZE,MAXBIT,MINBIT,ISEED)
      ID=1
      INDEX=0
      K=0
C
   15 CALL LEXSUB(N,K)
      POSTOT=0
      WRDTOT=0
      DO 30 L=1,K
      M=WRDSZE
      DO 20 J=1,WRDSZE
      IF(POSWDS(COMBIN(L),J).EQ.1) POSTOT=POSTOT+M
   20 M=M-1
   30 WRDTOT=WRDTOT+COMBIN(L)
      INDEX=INDEX+1
      PAIRS(INDEX,1)=POSTOT
      PAIRS(INDEX,2)=WRDTOT
      POSTOT=0
      WRDTOT=0
      L=1
      I=1
   40 IF(L.EQ.COMBIN(I)) GO TO 70
   50 M=WRDSZE
      DO 60 J=1,WRDSZE
      IF(NEGWDS(L,J).EQ.1) POSTOT=POSTOT+M
```

```
 60 M=M-1
    WRDTOT=WRDTOT+L
    IF(L.EQ.N) GO TO 110
    L=L+1
    GO TO 40
 70 IF(L.EQ.N) GO TO 110
    IF(I.GE.K) GO TO 80
    IF(I.EQ.N) GO TO 110
    I=I+1
    L=L+1
    GO TO 40
 80 L=L+1
    DO 100 I=L,N
    M=WRDSZE
    DO 90 J=1,WRDSZE
    IF(NEGWDS(I,J).EQ.1)  POSTOT=POSTOT+M
 90 M=M-1
100 WRDTOT=WRDTOT+I
110 PAIRS(INDEX,3)=POSTOT
    PAIRS(INDEX,4)=WRDTOT
    IF(K.NE.0) GO TO 15
    M1=(2**N)-1
    M2=M1-1
    LIMIT1=1
    LIMIT2=1000
    ITEMS=1000
140 ICOUNT=0
    ICOMP=0
    DO 120 I=1,M2
    K=I+1
    DO 130 J=K,M1
    IF(I.LT.LIMIT1) GO TO 130
    IF(J.GT.LIMIT2) GO TO 130
    ICOMP=ICOMP+1
    IF(PAIRS(I,1).NE.PAIRS(J,1)) GO TO 130
    IF(PAIRS(I,2).NE.PAIRS(J,2)) GO TO 130
    IF(PAIRS(I,3).NE.PAIRS(J,3)) GO TO 130
    IF(PAIRS(I,4).NE.PAIRS(J,4)) GO TO 130
    ICOUNT=ICOUNT+1
130 CONTINUE
120 CONTINUE
    COUNT=ICOUNT
    XCOMP=ICOMP
    X=(COUNT/XCOMP)*100.0
    WRITE(6,150) ITEMS,ICOMP,ICOUNT,X
150 FORMAT(' IN NEXT ',I5,' ITEMS THERE WERE ',I10,
   1' COMPARISONS WITH',I5,' DUPLICATES = ',F5.2,'%')
    LIMIT1=LIMIT1+1000
    LIMIT2=LIMIT2+1000
    IF(LIMIT2.LE.M1) GO TO 140
    GO TO (160,170,1),ID
160 ID=2
    LIMIT2=M1
    ITEMS=(LIMIT2-LIMIT1)+1
```

```
      GO TO 140
  170 ID=3
      LIMIT1=1
      LIMIT2=M1
      ITEMS=M1
      GO TO 140
  180 STOP
      END
      SUBROUTINE LEXSUB(N,K)
      INTEGER COMBIN
      COMMON COMBIN(15)
C
      IF(K.NE.0) GO TO 20
      IS=0
   10 IF(K.NE.N) K=K+1
   40 COMBIN(K)=IS+1
      RETURN
   20 IF(COMBIN(K).EQ.N) GO TO 30
      IS=COMBIN(K)
      GO TO 10
   30 K=K-1
      IF(K.EQ.0) RETURN
      IS=COMBIN(K)
      GO TO 40
      END
      SUBROUTINE BITSET(WORDS,N,WRDSZE,MAXBIT,MINBIT,ISEED)
      INTEGER WORDS,WRDSZE,BITPOS,BITCNT
      DIMENSION WORDS(15,20)
      X=WRDSZE
      DO 5 I=1,N
      BITCNT=0
      DO 10 J=1,MAXBIT
      Y=RAND(ISEED)
      BITPOS=(Y*X)+1.0
   10 WORDS(I,BITPOS)=1
   12 DO 15 K=1,WRDSZE
      IF(WORDS(I,K).EQ.1) BITCNT=BITCNT+1
   15 CONTINUE
      IF(BITCNT.GE.MINBIT) GO TO 5
      Y=RAND(ISEED)
      BITPOS=(Y*X)+1.0
      WORDS(I,BITPOS)=1
      GO TO 12
    5 CONTINUE
      RETURN
      END
      FUNCTION RAND(ISEED)
      INTEGER ICON/Z7FFFFFFF/
      IF(ISEED.EQ.0) ISEED=39
      ISEED=ISEED*23*13**4
      IF(ISEED.LT.0) ISEED=ISEED+ICON+1
      RAND=ISEED/FLOAT(ICON)
      RETURN
      END
```