CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

KEY DISTRIBUTION AND DATA INDEPENDANCE IN CRYPTOSYSTEMS

by

Shui-Mui (May) Wong

A thesis
presented to University of Ottawa
in partial fulfillment of the
requirements for the degree of
Master of Science
in
System Science
Department of Computer Science

OTTAWA, Ontario, 1983

University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

ABSTRACT

A cryptosystem makes use of a transformation and some keys. The security of the system thus depends on the secure management of the keys and the 'robustness' of the transformation.

In this thesis, we first describe some of the methods for cryptanalytic attacks, key distributions and digital signatures. We then modify some of the key distribution and digital signature techniques for possible application in a videotex environment (Chapter I and II).

Our major work is the proposal and computational experiments of a statistical technique for testing one of the properties about the 'robustness' of encryption algorithms (Chapter III). In fact, the Chi-square test is used to detect the functional independence between the plaintext and the ciphertext (for a fixed key), and between the key and the ciphertext (for a fixed plaintext) of a cryptosystem.

Though the method can be applied to a general encryption algorithm, we do our computational experiments for DES (Data Encryption Standard Algorithm of National Bureau of Standards) because of its widespread acceptance. A lot of efforts have been spent in the coding (Chapter IV).

42 tests have been performed, 21 for testing the functional independence between the plaintext and the ciphertext and 21 for testing the key and the ciphertext. Our test results show that independence between these data can be assumed for DES.

ACKNOWLEDGEMENTS

The author wishes to express her deepest gratitude to her thesis supervisor Dr. T.Y. Cheung for his constant advice, valuable suggestions and guidance. He always found time for productive discussions and encouragement to her work.

She also would like to thank the help of all her professors and colleaques, especially Dr. R.L. Probert, Dr. L.G. Birta, Dr. G. White, and Mr. P. Hickey.

The invaluable help of the university staff, especially those of the Department of Computer Science, is also appreciated.

The financial supports from the University of Ottawa, National Science and Engineering Research Council of Canada and the Department of Communications are acknowledged with gratitude.

Lastly, but not the least, the author is indebted to the very special supports she received from her parents Mr. & Mrs. P.C. Wong, and her friends Mr. S.B. Chang and Miss Y.M. Cheung.

CONTENTS

Chapter I

FUNDAMENTALS OF DATA ENCRYPTION AND VIDEOTEKT SYSTEMS

1.1 INTRODUCTION

Data encryption depends mainly on mathematical transformations. Old-fashioned cryptosystems rely heavily on keeping these transformations secret. In modern cryptography, the transformations are often known to the public and security of a cryptosystem depends on keeping the keys secret. Two different approaches for handling the keys result in what we now generally accept as the conventional cryptosystems and public-key cryptosystems (DAVI82, DIFF76, RIVE78, NEED78, POPE79, DIFF79, MERK78, DENN81). In this Chapter, we explain the fundamental concepts of these cryptosystems and some cryptanalytic schemes of attacking them. Some relevant applications will be reviewed in Chapter II.

Since we shall consider some encryption-based security problems for videotex systems, a brief review of some major videotex systems is included.

1.2 FUNDAMENTALS OF OLD-FASHIONED, CONVENTIONAL AND PUBLIC-KEY CRYPTOSYSTEMS

i) Old-fashioned cryptosystems:

Historically, encryption methods have been divided into two categories: substitutions and transpositions.

a) Substitution

In this category of methods, each letter or group of letters of the plaintext is replaced by another letter or group of letters according to a certain rule. The result is the ciphertext (TANE81).

A particular approach is the method of permutation on the symbols of the plaintext, as shown in the following

Plaintext:

a b c d e f g h i j k l m

n o p q r s t u v w x y z

ciphertext:

q w e r t y u i o p a s d

f g h j k l z x c v b n m

By this permutation, the word 'attack', for instance, will be encrypted as 'qzzqea'. This method is called a monoalphabetic substitution.

A method in which every letter in the plaintext is replaced with two letters is called a 2-alphabetic substitution. A more complex method, called polyalphabetic substitution, can be obtained by introducing the concept of key and table lookup (HARR73).

b) Transposition

In this category of methods, the ciphertext is produced by reordering the letters of the plaintext (LEMP79).

One of the commonly used transposition methods is called columnar transposition. In this method, a keyword without repeated letters (e.g., MEGABUCK) is used to index alphabetically the columns of a table. The length of the keyword determines the number of columns of the table. The plaintext is written horizontally as a series of rows. The ciphertext is formed by reading the columns of the table, starting with the column under the (alphabetically) lowest letter, then the column under the second lowest, and so on. This is illustrated by the following example:

```
                    M E G A B U C K     <---- Keyword
                    - - - - - - - -
Plaintext-->        p l e a s e t r     Ciphertext
                    a n s f e r o n
                    e m i l l i o n       |
                    d o l l a r s t       |
                    o m y s w i s s       ↓
                    b a n k a c c o
                    u n t s i x t w
                    o t w o a b c d
```

Plaintext : pleasetransferonemilliondollarst
omyswissbankaccountsixtwotwoabcd

Ciphertext: afllsksoselawaiatoossctclnmomant
esilyntwrnntsowdpaedobuoeriricxb

The keyword in this example is 'MEGABUCK'. The plaintext is written row by row under the keyword. To form the ciphertext, the column under the letter 'A' of the keyword is read first, and then the columns under 'B', 'C', 'E' and so on.

ii) Conventional cryptosystems:

A conventional encryption process makes use of a mathematical transformation F, and a key SK, such that

$$C = F(P,SK)$$

where P is the plaintext (the message to be encrypted), and C is the ciphertext (the encrypted message). For F to be useful, there must exist another transformation F', the inverse of F, for recovering P from C, i.e.,

$$P = F'(C,SK).$$

Usually, both F and F' are known to the public and use the same key SK, and the key SK is kept secret. Thus, this approach is useful only if it is very difficult to recover P from C without the knowledge of the key SK.

The DES (Data Encryption Standard) Algorithm is the most well-known methmatical transformation used in conventional cryptosystems (SMID81, KENT81, BRAN76, GAIT77b, HEAT76, MORR77, STEP78, SYKE77, TUCH78, YASA76). It was developed by IBM and adopted by NBS (National Bureau of Standards) as a USA standard (NBS77, BRYC81). The algorithm consists of a sequence of product transformations (i.e., combinations of transpositions and substitutions). Its details will be described in Chapter IV.

iii) Public-key cryptosystems:

In 1976, Diffie and Hellman (DIFF76) proposed a new direction for cryptographic developments (MERK78, RIVE76c, RIVE79, SIMM77, PRIC81, SIMM79, MERK78b, SHAM80, POHL78, ADLE79, HERL78, NEED78, DAVI80b). In their approach, a transformation F is used for encrypting the data P

$$C = F(P,PK)$$

while another transformation F´

$$P = F´(C,SK)$$

is used for recovering the original data P. The decryption key SK (also called the secret key) is different from the encryption key PK (also called the public key) and is difficult to be derived from PK. In practice, the two transformations and the encryption key PK are known to the public, while the decryption key SK is kept secret. Obviously, there must be some relationship between the two keys and between the encryption and decryption transformations.

Well-known mathematical transformations for the public key cryptosystems are the MIT Algorithm proposed by Rivest, Shamir, and Adleman (RIVE78) and the Trapdoor Knapsack Algorithm proposed by Merkle (MERK78).

1.3 CRYPTANALYSIS

Cryptanalysis is the subject in which we study the methods for breaking a cryptosystem. (DIFF77, GARD77, SINK68, DAVI79, HELL79, KAHN76, KULL76, TUCH79, RIVE78, RIVE79, HOFF77, BRIG77, EDWA66, GROS74, TUCK73, HELL76, BARK77, FRIE44, GAIT56). In old-fashioned cryptography the aim is to discover the mathematical transformations. For modern cryptosystems the aim is either to discover the keys or to recover the plaintexts directly. Three approaches are commonly used for cryptanalytic attacks: ciphertext only attacks, known plaintext attacks and chosen plaintext attacks:

i) . Ciphertext only attack

A ciphertext only attack is a cryptanalytic process in which the cryptanalyst has obtained a large amount of ciphertext and the attack is based almost entirely on the information about them (TANE81,HOFF77,BRIG77). If the cryptanalyst also has some general (but not specific) knowledge (e.g., the format) about the plaintext, he may be able to crack the system more easily. For example, he may know that the plaintext is a Telidon data stream, but does not know exactly what its contents are.

In the following we describe a typical procedure of how a cryptanalyst attacks an old-fashioned cryptosystem.

A cryptanalyst first collects a substantial amount of ciphertexts. To determine if they are generated by the method of transposition, he can make use of certain statistical properties of the plaintext. For example, if the plaintext is an English message, statistical information about the occurrence frequencies of English letters is available. (e.g., E, 13.05%; Z, 0.09%; digrams TH, 3.16%; RO, 0.55%; trigrams ING, 1.42%; CON, 0.45%; words THE, 6.42%; NOT, 0.61%; etc.) (TANE81). He then counts the frequencies of the letters appearing in the collection of ciphertexts and compares these frequencies with those of the English letters. If they have a similar frequency distribution, those ciphertexts are generated by the method of transposition since a transposition simply reorders the letters and does not change their occurrence frequencies.

If the frequency distributions are different, the collected ciphetexts may be generated by the method of substitution. In order to find out whether this is true, a cryptanalyst can calculate the index of coincidence, IC, of the ciphertext. It is defined as

$$IC = \frac{\sum_{i=A}^{Z} f_i (f_i - 1)}{N(N-1)}$$

where f is the number of occurrences in the ciphertext of the i letter of the alphabet, and N is the number of letters in the whole ciphertext.

If the plaintext is in English and the IC value is greater than 0.066, probably monoalphabetic substitution has been used. If the IC value is between 0.052 and 0.066, a 2-alphabetic substitution is possible. If the IC value is less than 0.038, a polyalphabetic substitution is expected.

If the IC value indicates that the ciphertext may have been generated by the method of monoalphabetic substitution, a cryptanalyst can use the following technique to attack the ciphertext. He starts by counting the relative frequencies of all the letters in the ciphertext. Then, he assigns the most common one to 'e' and the next common one to 't'. After the substitutions, if there is a pattern, e.g., 'tXe', in the ciphertext, he looks at the trigrams to find a common one of the form 'tXe'. It suggests that 'X' is 'h'. Similarly, if the pattern 'thYt' occurs frequently, the 'Y' probably stands for 'a'. With this information, he can look for a frequently occuring trigram of the form 'aZW' which is most likely 'and'. By making guesses of the commonest letters,

digrams, and trigrams, the cryptanalyst builds up a tentative plaintext.

In some cases, a cryptanalyst may also make use of certain knowledge of some commonly used phrases to build his plaintext. For example, a business letter most probably begins with 'Dear Sir:' and many time-sharing computer systems begin with a message 'Please logon'.

Consider the following ciphertext created by an accounting firm (grouped into blocks of five characters each):

CTBMN BYCTC BTJDS QXBNS GSTJC BTSWX CTQTZ CQVUJ

QJSGS TJQZZ MNQJS VLNSX VSZJU JDSTS JQUUS JUBXJ

JDSKS UJSNT KBGAQ JZBGY QTLCT ZBNYB NQJSW ABCDE

A word which occurs frequently in an accounting firm report is 'financial'. Using the knowledge that 'financial' has a repeated letter 'i', with four other letters between their occurrences, a cryptanalyst looks for repeated letters with this spacing in the ciphertext. He finds 10 such cases starting at positions 6, 15, 27, 31, 42, 48, 56, 70, 71, and 83. However, only two of these groups, those starting at positions 31 and 42 respectively, have the second letter (the letter 'n' in the

plaintext) repeated in the proper places. Furthermore, among these two, only the first has the letter 'a' correctly positioned. Thus he knows that the word 'financial' begins at position 30.

Ciphertext only attack is the weakest threat a cryptosystem is usually subject to. Any system which succumbs to it must be considered totally unsecure.

ii) Known plaintext attack and chosen plaintext attack

A known plaintext attack is a cryptanalytic process in which the cryptanalyst has knowledge of a substantial amount of released plaintexts and their corresponding ciphertexts (TANE81, BRIG77). For a modern cryptosystem the goal of the attack is to discover the key for later usage by making use of such knowledge. For example, commercial product announcements or press releases may first be sent in encrypted forms and disclosed to the public later. A cryptanalyst can first intercept the ciphertext and later match it with the released plaintext.

In a known plaintext attack, one possible approach is for a cryptanalyst to first collect a possible list of keys, e.g., names of babies, names of streets, etc. He then encrypts the known plaintext with each of these keys and compares the

resulting ciphertext with the known ciphertext. If they are the same (or in similar patterns), the key is trapped. The drawback of this approach is that it is very time consuming to try all the possible keys.

In order to save encryption time, a cryptanalyst can encrypt some common plaintexts (e.g., PLEASE LOGON, DEAR SIR, etc.) in advance by using all the keys he has collected. He then stores the pairs of keys and ciphertexts. Later, if he wants to break the ciphertext, he simply locates the same ciphertext in the pairs of keys and ciphertexts which he has already stored. He then uses the extracted encryption key to decrypt the ciphertext to get the corresponding plaintext. The drawback of this approach is that it requires a large storage area for all the keys and ciphertexts.

The trade-off of these two approaches is time and computer memory.

A chosen plaintext attack is similar to the known plaintext attack. The difference is that in a known plaintext attack the plaintext is always obtained from the user of the system; whereas in the chosen plaintext attack, a cryptanalyst plans and creates his own plaintext which can help him to obtain knowledge about the structure of the keys,

etc. He then traps the system to encrypt it and intercepts the ciphertext.

A system that succumbs to known or chosen plaintext attack is considered unsecure.

1.4 VIDEOTEXT SYSTEMS

i) Videotext services

A videotex system has three subsystems: the user terminals, the information providers' terminals, and the information centre. A user terminal consists of a display monitor, a microprocessor and a keypad (see Figure 1.1). A user selects data for display on the screen by using menu-lookup searching techniques.

Videotex service may be one-way or two-way. One-way service is also known as teletext. In such a system all the information pages are continuously broadcast in cycles to all the users. A user cah select those pages he wants. Ceefax (WOOL80) is an example of such systems. Two-way service is also known as viewdata. It needs a two-way connection between the user terminal and the information centre by telephone lines, coaxal cable, etc. Examples include England's Prestel System (TROU80) and France's Antiope system (MART79).

Following are a few major videotex systems:

Bell Canada's Vista

Vista (WOOL80) is a videotex system having a single host. All users and information providers are connected to the host.

Britain's Prestel

Britain's early Prestel System (TROU80) consisted of several information retrieval computers (IRC) and one update centre (UPC). Each of the IRC's contained an identical database. The current plan is to include three update centres.

France's Antiope

In France's Antiope system (MART79), the videotex centres, databases, and update centres are connected by the Transpac Data Network. Users are connected to the information retrieval computers by telephone lines. A particular unit called DIDON provides the broadcast services.

ii) Data encryption in videotex systems

Data streams in certain videotex systems have special formats. For example, a Telidon system uses the PDI (Picture Description Instruction) opcodes (OBRI82) in coding the data. There are 32 PDI opcodes. Since they are publicly known, data encryption of Telidon data will be particularly

sensitive to cryptanalytic attacks (see Section 1.3).

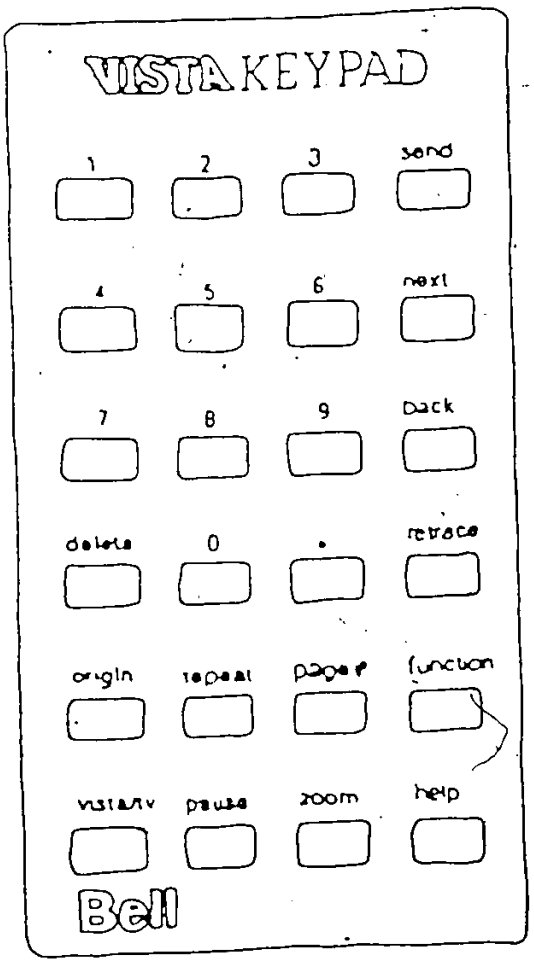
A 'conventional' (i.e., non-integrated) videotex system has the following limitations:

a) User terminals have very little processing capability.

b) For teletext systems (one-way broadcast) no feedback is possible.

c) For a videotex system using a terminal with a simple keypad, the input tool is limited to a few numeric keys and special function keys.

The above limitations restrict the applicability of data encryption techniques to a 'conventional' videotex system. A cryptosystem requires more intelligence and computing facilities in the user terminals in order to distribute the communication keys and encrypt or decrypt the messages. An ordinary user terminal in a 'conventional' videotex system cannot handle these problems.



LABELS

send - execute operation

next - next page

back - previous page

delete - erase previous character

retrace - repeat in reverse order last ten pages viewed

origin - restore service directory to screen

repeat - display page again

page # - display page number

function- followed by other key(s), specifies operations

vista/tv- sign-on/sign-off

pause - halt display until pressed again

zoom - expands top or bottom half of page

help - requests user instructions

Figure 1.1 Keypad for the VISTA System.

Chapter II

KEY MANAGEMENT AND DIGITAL SIGNATURES

2.1 INTRODUCTION

In designing a cryptosystem for a network environment, one of the major issues is to properly distribute the keys to the senders and receivers. The keys should not be permanent and must be changed when there is fear of being compromised. Keys must be provided to new users and old keys be discarded as users withdraw. Consideration of all these problems forms the subject of key management (MATY78, EHRS78, SEND78).

In business, authenticity of documents for legal, or financial purposes is determined by the presence of an authorized handwritten signature. However, handwritten signatures cannot be used in a modern information system because they cannot be computerized and automatically transmitted to the users. Digital signatures which are devised to replace the handwritten ones can be used to authenticate these documents (FEIS58, NEED78, WIDM76, RIVE78a, DAVI80a, BOOT81).

In this Chapter, we shall discuss the solutions for some of the problems related to key distributions and digital signatures. Section 2.2 presents a key distribution protocol

for conventional cryptosystems. Section 2.3 reviews Merkle's method for key distribution. Section 2.4 discusses a key distribution protocol for public-key cryptosystems. Section 2.5 reviews a key distribution protocol using the exponential function in modulo arithematic. Section 2.6 discusses briefly some public-key digital signature protocols. Section 2.7 proposes some modifications on the conventional-key and the public-key distribution protocols for a videotex system. Section 2.8 studies the application of the exponential-function-based key distribution protocol to a videotex system.

2.2 A CONVENTIONAL-KEY DISTRIBUTION PROTOCOL

In a conventional cryptosystem, both the sender and the receiver use the same key. For each pair of sender and receiver, two identical keys are created and delivered personally between themselves. This key distribution method is inconvenient and insecure, especially for those cryptosystems which have many users or have to change the keys frequently. In order to communicate securely with the other users, one has to maintain all their keys. If the cryptosystem is large, secure storage of these keys will be a difficult problem.

In the following we describe a key distribution protocol for conventional cryptosystems (NEED78, PRIC81, GUDE80, KENT81, SIMM79). In this protocol, a KDC (key distribution

centre) will handle the creation and transmission of keys. For each user, two keys are involved. A key, called the communication key, is temporarily created for each communication session. It is discarded once the communication is over. Another key, called the secret key, is known only to the KDC and the user. It is used for encrypting the communication keys.

Notation: $(X,Y)^{K_i}$ means that the concatenation of X and Y is encrypted with the key K_i of user i.

Assume that user A initiates a request for communication with user B. The following five steps (see Figure 2.1) describe how a communication key can be created and transmitted to the two users, A and B.

(1) To establish a connection, user A sends a request to the KDC. The request contains his own identity A and a message $(R,B)^{SK_a}$. This message contains a random number R and the identity of B and is encrypted with the secret key of A.

A->KDC: $A,(R,B)^{SK_a}$

(2) The KDC obtains R by decrypting the encrypted part of A's message. He then sends the following encrypted information to A: the communication key K_c, the random number R, and the message $(K_c,A)^{SK_b}$. The random number R is included in the reply so as to prevent an intruder from impersonating the KDC by

inserting a previous communication key which the intruder wants users A and B to use again.

KDC->A: $(K_c, R, (K_c, A)^{SK_b})^{SK_a}$

(3) When user A receives this reply, he decrypts it with his secret key. He extracts the encrypted portion of the incoming message, $(K_c, A)^{SK_b}$, and sends it to user B together with a request identity, CR.

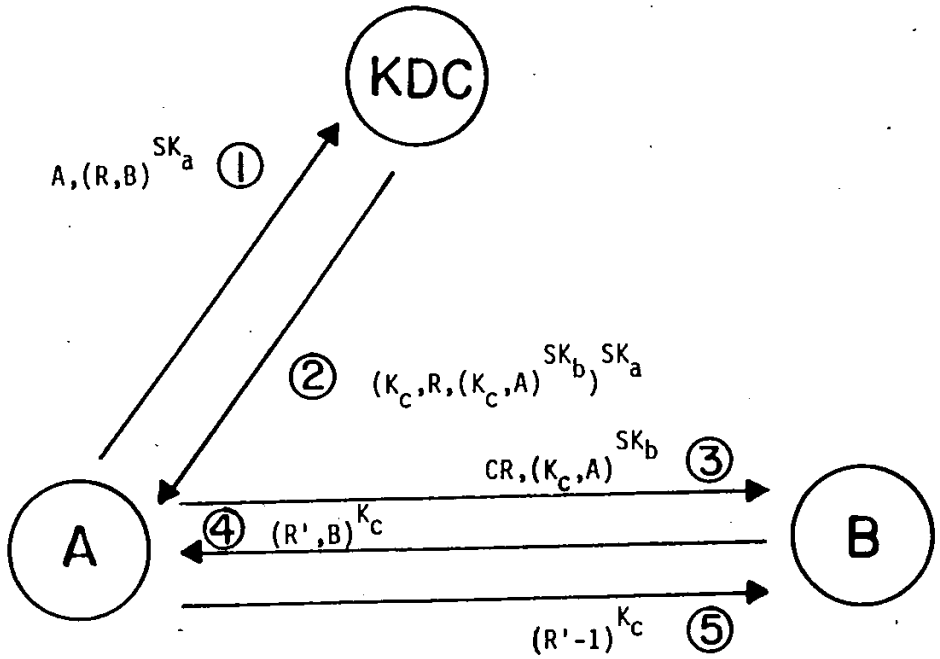
A->B: $CR, (K_c, A)^{SK_b}$

(4) Now only user B can get the communication key K_c, since it is encrypted with user B's secret key. User B may need further assurance that it is indeed user A who sends the request and not some intruder using request of the previous communication. Thus, user B includes in his reply to user A a different random number R', and his own identity, encrypted with the communication key, K_c.

B->A: $(R', B)^{K_c}$

(5) After checking user B's identity, user A modifies the random number R' in a prearranged manner (e.g., changing R' to R'-1). He then encrypts and returns the modified number to user B. In this way, user A authenticates himself to user B.

A->B: $(R'-1)^{K_c}$



A,B - Users

KDC - Key Distribution Centre

SK_a - Secret Key of A

SK_b - Secret Key of B

K_c - Communication Key

$(\ )^K$ - Encryption under K

CR - Call Request

R,R' - Random numbers

Figure 2.1 A conventional key distribution protocol (PRIC81).

After these steps, user A and user B have authenticated their own identities and can start their data communication.

2.3 MERKLE'S METHOD FOR KEY DISTRIBUTION

Merkle (MERK78a) proposed another key distribution method for conventional cryptosystems. The method is based on the concept of a 'puzzle'. A 'puzzle' is defined by Merkle as a cryptogram (also called ciphertext) intended to be broken. This method is described below in the context of the DES algorithm (TANE81).

Two users A and B wish to communicate in a secure way. Suppose that user A initiates the conversation. His first message to user B may read as follows:

"Dear B,

I am sending you 20,000 puzzles. Each puzzle is a cryptogram whose corresponding plaintext starts with 128 zero bits and is followed by a 16-bit puzzle number and then a 56-bit key. These cryptograms have been encrypted using the DES algorithm with a key of 56 bits long whose last 22 bits are all zero. Please, pick one of these cryptograms randomly and break it by brute force, For example, by trying all the 2**34 56-bit keys, each ending with 22 zeros. On finding a key that yields a plaintext starting with 128 zeros, you will have broken the cryptogram. Then extract the 56-bit number at the end and use it as your key for our communication. As your

first message, send me back the puzzle number in an unencrypted form, so that I know which key you are going to use.

Your secretive correspondent, A"

The above message is followed by 20,000 puzzles. User B then selects one of them at random, decrypts it according to user A's instruction, and obtains the puzzle number and the communication key. User B will send the puzzle number to user A. When user A receives it, he can look up for its corresponding key. This key will be used to encrypt and decrypt messages in both directions throughout the communication.

The main point of this method is that a user has only to break one puzzle by brute force, while an intruder has to try an average of 10,000 puzzles in order to compromise the key.

2.4 PUBLIC-KEY DISTRIBUTION PROTOCOLS

Theoretically, the algorithm and the public key of a public-key cryptosystem can be made public (see Section 1.2). It seems that this solves the problems of key distribution. However, this is not so. The security of a public-key cryptosystem depends on the correct public-key being selected by the sender. If the key listed in a public directory is a wrong one, (probably altered by an intruder),

the purpose of using a public key is lost. Also, the users themselves may sometimes want to change the public keys for fear that their secret keys have been compromised.

Thus, a secure public-key cryptosystem requires the use of an up-to-date public key. One way to distribute and maintain the up-to-date public keys is by means of a key distribution centre. The following protocol (see Figure 2.2) describes how such keys can be transmitted to two users A and B from a key distribution centre (KDC) (PRIC81, SMIM79). It is assumed that user A initiates the communication request.

(1) User A requests the public key of user B by sending the following message to the KDC, which contains the identities of both A and B.

A->KDC: (A,B)

(2) The KDC sends to user A the public key of user B, and a message, $(PK_b,A)^{SK_{KDC}}$, all encrypted with the secret key SK_{KDC} of the KDC.

KDC->A: $(PK_b,(PK_a,A)^{SK_{KDC}})^{SK_{KDC}}$

(3) After decrypting this message with the public key of KDC, user A extracts the public key of user B and the encrypted message $(PK_a,A)^{SK_{KDC}}$. He then sends a communication request CR and the encrypted message $(PK_a,A)^{SK_{KDC}}$ to user B.

A->B: $CR, (PK_a, A)^{SK_{KDC}}$

(4) On recognizing the command CR, user B knows that the received messageis a communication request. He then decrypts the encrypted part of the message with the public key of KDC. He now knows user A is the sender and his up-to-date public key is PK_a. To prove that the message is not sent by an intruder, user B sends to user A a random number R, encrypted with the public key of user A.

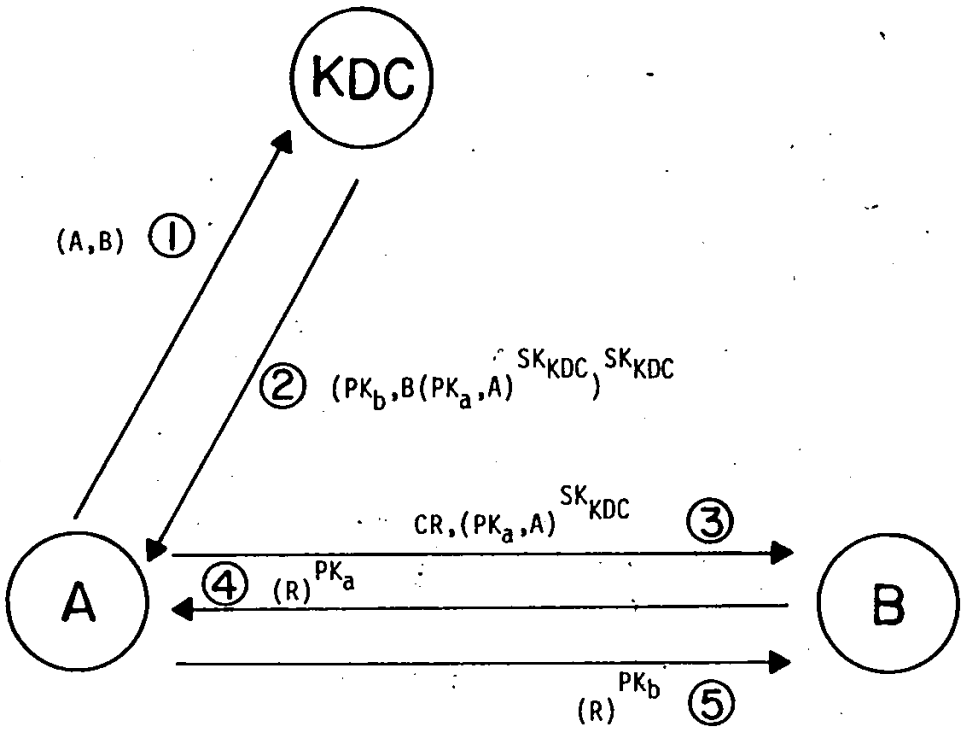
B->A: $(R)^{PK_a}$

(5) User A identifies himself to user B as follows: He first obtains R by decrypting the received message and re-encrypts R with the public key of user B. He then sends the re-encrypted R to user B.

A->B: $(R)^{PK_b}$

(6) On receiving this message, user B decrypts it with his own secret key and obtains R. He compares this R with the one he sent to user A previously. The messages sent by A are valid if they are the same, because only A can have obtained R by decrypting the message in (4).

After these steps, users A and B have obtained the up-to-date public keys and have authenticated their own identities. They can start their data communication.



SK_{KDC} - Secret Key of KDC

PK_a - Public Key of A

PK_b - Public Key of B

$(\ )^K$ - Encryption under K

R - Random number

CR - Call Request

Figure 2.2 A public key distribution protocol (PRIC81).

2.5 AN EXPONENTIAL-FUNCTION-BASED KEY DISTRIBUTION PROTOCOL

This protocol makes use of the following property of the exponential function in modulo arithmetic.

For a given prime number, m, and appropriate integers a and x (e.g., when a and m are relatively prime, x between 0 to m-1), the exponential function, $b=a^x \pmod m$, is a one-to-one mapping onto the set $\{0,1,\ldots,m-1\}$. As a result, b has an inverse, $x=\log_a b \pmod m$. For a given value of b, it has been proven (DAVI82) that, in most cases, it is very difficult to calculate x from this inverse function when m is large. Thus, $b=a^x \pmod m$ is a one-way function. This property is used in the following key distribution protocol.

Assume that a and m are publicly known. Users A and B choose secretly the integers x and y, respectively, from the range of 0 to m-1. A computes the value $a^x \pmod m$ and sends it to B. Similarly, B computes the value $a^y \pmod m$ and sends it to A. Knowing x and $a^y \pmod m$. A can now compute $(a^y)^x \pmod m$, which is the same as $a^{xy} \pmod m$. Similarly, B can compute $(a^x)^y \pmod m$ which is also $a^{xy} \pmod m$.

The value of $a^{xy} \pmod m$ will be used as a communication key throughout the communication session between users A and B.

An intruder knows only the values of a, m, $a^x \pmod m$ and $a^y \pmod m$. In order to compromise the communication key, he has to invert the exponential function $a^x \pmod m$ or $a^y \pmod m$ so as to obtain the value of x or y. This process can be

made impractical by choosing an extremely large prime number m (DAVI82).

2.6 DIGITAL SIGNATURE PROTOCOLS

A digital signature protocol has three parts: a method for signing a message, a method for authenticating a signature, and a method for resolving disputes. Some digital signature protocols are briefly described below:

i) A digital signature for public-key cryptosystems

The use of public-key cryptosystems for providing digital signatures was first suggested by Diffie and Hellman (DIFF76). It is based on the fact that encrypting the decrypted form of a message m yields m itself, i.e., m=E(D(m,SK),PK).

User A signs a message m by decrypting it with his own secret key, SK. He sends the decrypted message D(m,SK) to user B as a signed message. User B can compute E(D(m,SK),PK) (i.e., encrypting the decrypted message with the public key of A) and get back the original message m.

The above procedures can confirm that user A signed the message because only he has the decryption key.

ii) A time-stamped digital signature for public-key cryptosystem

It is often desirable to be able to solve the following problems involving digital signatures:

(1) After the reported loss of a key, to deny the issuance of any messages signed by someone else.

(2) To prevent a user from denying his signatures which are signed before a reported key loss.

The following time-stamp protocol was proposed by Merkle (MERK80) to answer those problems. In his approach, he assumes that users A and B agree on using a time-keeper to indicate the time that a message is received. User, A, signs a message, m, by decrypting m with his secret key, and sends the decrypted message, D(m,SK), to user B. User B then uses a time-keeper to time-stamp the message and asks the central authority (CA) to check if user A's secret key is still valid at the current time. The CA signs a message to confirm the status of A's secret key and sends it to user B. If user A's secret key has been reported lost, user B rejects the signature, otherwise he accepts it.

The time-stamp together with the message signed by the CA can confirm that user A is held responsible for sending the message, since only user A has the secret key and the time-stamp can

verify that the message was signed before the reported loss of his secret key.

2.7 MODIFICATION OF SOME EXISTING KEY DISTRIBUTION PROTOCOLS FOR A VIDEOTEX SYSTEM

In designing a key distribution protocol for a 'conventional' videotex system, two properties should be taken into consideration: Firstly, a 'conventional' videotex system is essentially for information retrieval. Most data move in one direction from the Information Retrieval Centre (IRC) to the users. There is no communication among the users themselves. That is, during each communication session, only one user participates. Thus, the IRC can maintain a central control of the whole system and act as the Key Distribution Centre (KDC) at the same time. Secondly, the user terminals of a 'conventional' videotex system have very limited processing and operational capabilities. Their keypad has only a few function and character keys and the users are mostly non-technical people. The burden of communication key creation, etc. lies on the KDC. In order to implement a cryptosystem in a videotex system, a terminal should possess at least the ability of decrypting encrypted data and handling key management. Because of the real-time environment, software implementation of such algorithms will be very inefficient. LSI chip implementation seems to be the natural approach. A chip for the DES Algorithm is being manufactured (DATA82)

and another for the MIT Algorithm (RIVE78) is being designed in Japan (MIYA82).

With the above features in mind, we modify the conventional key distribution protocol and the public-key distribution protocol of Sections 2.2 and 2.4 respectively, so that they will be more suitable for a 'conventional' videotex system.

i) A modified protocol for conventional-key distribution in a videotex system

We modify the conventional-key distribution protocol of Section 2.2 as follows: (see Figure 2.3). The IRC keeps a secret key for every individual user.

(1) To initiate a request, user A encrypts a random number R with his secret key SK_a and sends it together with his identity A to the IRC.

A-->IRC: $A, (R)^{SK_a}$

(2) IRC decrypts the message with A's secret key and gets the random number R. He then encrypts a communication key K_c and R with the secret key of A. The encrypted message is sent to A. R is included in the message so that A knows that the communication key really comes from IRC, because only the IRC can have decrypted A's previous message.

IRC-->A: $(K_c, R)^{SK_a}$

(3) A identifies himself to the IRC by modifying the R in an agreed manner. This modified R is encrypted with the communication key K_c, and sent to the IRC.

A-->IRC: $(R-1)^{K_c}$

After these steps, A can start retrieving information from the IRC with the common communication key K_c.

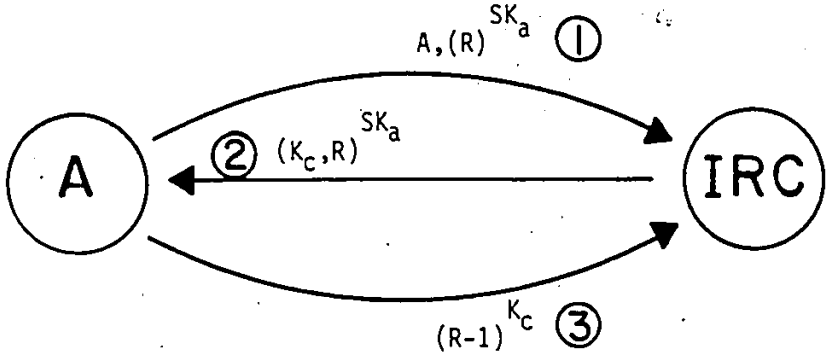


Figure 2.3 A modified protocol for conventional key distribution in a videotex system.

A - User

SK_a - Secret Key of A

K_c - Communication Key

R - Random number

IRC - Information Retrieval Centre

$( )^K$ - Encryption under K

ii) A modified protocol for public-key distribution in a videotex system

For a public-key cryptosystem, the IRC does not have the burden of keeping the users' keys secret. Both IRC and the users use publicly-known keys. Following is a modification of the public-key distribution protocol of Section 2.4 (see Figure 2.4):

(1) To initiate a request, user A encrypts a random number R with the public key PK_{IRC} of the IRC and sends it together with his identity A to the IRC.

A-->IRC $\quad A, (R)^{PK_{IRC}}$

(2) The IRC decrypts the received message with his secret key and gets the random number R. Then he modifies R in an agreed manner (e.g., changing R to R-1), encrypts a communication key K_c, the modified R, and another random number R' with the public key PK_a of A and sends the encrypted message to A.

IRC-->A: $\quad (K_c, R-1, R')^{PK_a}$

(3) A decrypts the reply with his own secret key. He knows that the communication key, K_c

, is from the IRC by checking the modified R. To identify himself to the IRC, he modifies the R' in an agreed manner (e.g., changing R' to $R'-1$), encrypts it with the communication key K_c and sends it to the IRC.

A-->IRC $(R'-1)^{K_c}$

After these steps, A can start retrieving information from the IRC with the common communication key K_c.

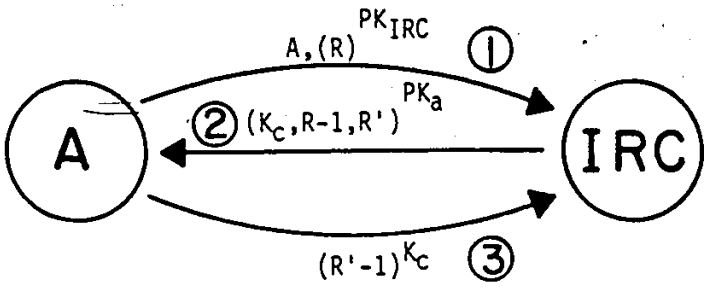


Figure 2.4 A modified protocol for public key distribution in a videotex system.

A - User

IRC - Information Retrieval Centre

PK_{IRC} - Public Key of IRC

PK_a - Public Key of A

R,R' - Random numbers

K_c - Communication Key

$( )^K$ - Encryption under K

2.8 APPLICATION OF THE EXPONENTIAL-FUNCTION-BASED KEY DISTRIBUTION PROTOCOL FOR THE VIDEOTEX SYSTEMS

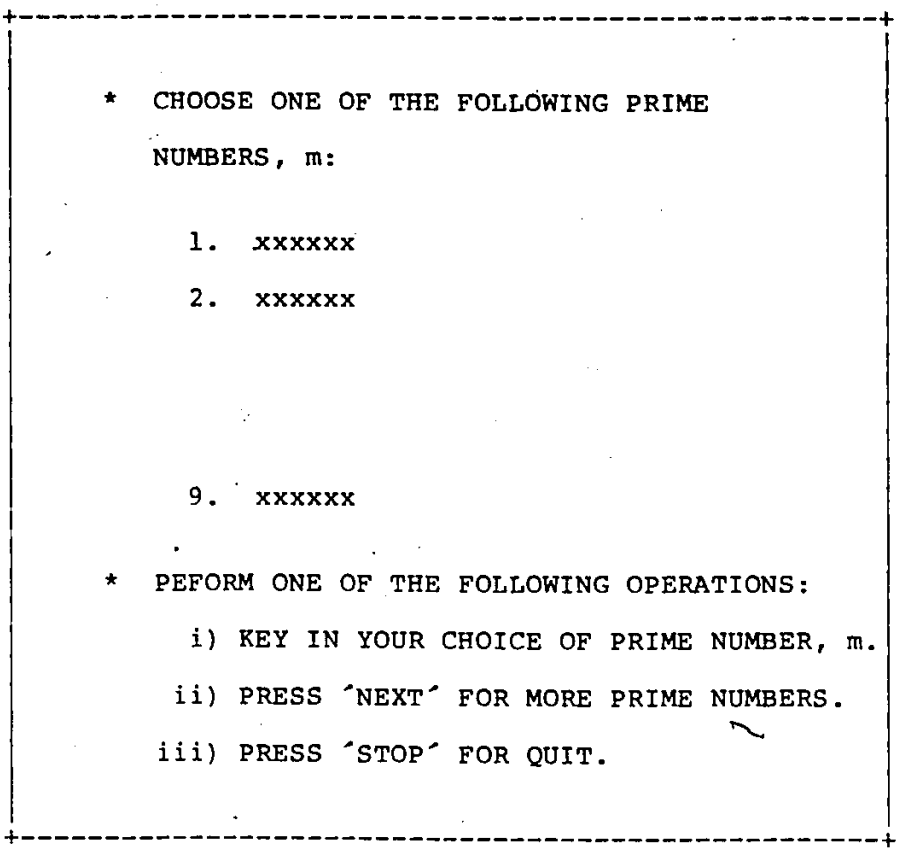
As mentioned in Section 2.7, a 'conventional' videotex terminal has very limited data processing capability and its users can only do simple searching, such as menu-lookup, etc. In this Section, we show how to execute the exponential-function-based key distribution protocol at such a videotex terminal without greatly expanding the capabilities of its keypad and CPU.

It is assumed that an LSI chip with the capability of evaluating an integral exponential function in modulo arithmetic and storing a few numbers is attached to the user terminal. Functions that are not available in the keypad can be operated by a combination of existing function keys and character keys. For example, pressing a key 'Origin' may mean 'to store the keypad input in the chip'; pressing the keys 'Origin' and 'Send' may mean 'to store the input in the chip and also send it to the IRC'.

A conversation between a user and the IRC for generating a communication key may run as follows:

(1) The user first initiates a key generation request by pressing a combination of certain function keys and character keys.

(2) On receiving the request, the IRC sends the following page of information to the user for guiding him to form a communication key.



(3) Suppose the user chooses Operation i), he presses the keys 'Origin', keys in one of these prime numbers and presses the key 'Send'. The number will be stored in the chip and also sent to the IRC automatically.

(4) On receiving the prime number m, the IRC sends to the terminal another page of prime numbers which are all relatively prime to m.

```
+-------------------------------------------------+
|                                                 |
|   * CHOOSE ONE OF THE FOLLOWING PRIME           |
|     NUMBERS, a.                                 |
|                                                 |
|       1.  xxxxxx                                |
|       2.  xxxxxx                                |
|                                                 |
|                                                 |
|       9.  xxxxxx                                |
|                                                 |
|   * PERFORM ONE OF THE FOLLOWING OPERATIONS:    |
|       i) DO THE FOLLOWING;                      |
|            1. KEY IN YOUR CHOICE OF PRIME       |
|               NUMBERS, a,                       |
|            2. PRESS THE 'BLANK' KEY,            |
|            3. KEY IN ANY POSITIVE INTEGER x.    |
|      ii) PRESS 'NEXT' FOR MORE PRIME NUMBERS.   |
|     iii) PRESS 'STOP' FOR QUIT.                 |
|                                                 |
+-------------------------------------------------+
```

(5) Suppose Operation i) is chosen. The chip will store the integers a and x and compute the value of $a^x \pmod m$ automatically. The numbers a and $a^x \pmod m$ will be sent to the IRC when the user presses the key 'Send'.

(6) On receiving these numbers, the IRC forms the communication key $a^{xy} \pmod m$ by using the received number $a^x \pmod m$ and a positive integer y chosen

randomly. It also computes the value of $a^y \pmod m$ and sends it back to the user.

(7) Using the received value $a^y \pmod m$ and the stored number x, the communication key $a^{xy} \pmod m$ is automatically computed by the chip. The terminal is ready to decrypt any information sent from the IRC and encrypted with the new communication key.

The above method is rather straightforward. Its drawback is that the user and the IRC are not sure that the exchanged messages come from the valid sender. There is no prior identification between them. To compensate for this, we shall describe in the following a time-stamped digital signature technique as a procedure for the user and the IRC to authenticate themselves.

Time-stamped key distribution protocol for videotex systems

The following time-stamped protocol is created by combining the techniques of the exponential-function-based key distribution protocol of Section 2.5, the public-key digital signature of Section 2.6, and the modified public-key distribution protocol of Section 2.7(ii).

Unlike Denning (DENN81) who used time-stamps as a means for validating the time integrity for two-way communications, a time-stamp is used here only as a 'password' for user identification. This is the terminology

used by Needham (NEED78, BOOT81, MERK80) who used it for one-way communications.

A time-stamp T is a piece of digital information containing a time-keeper and a sequence number. It is assumed that the user and the IRC have agreed on a specific time-keeper. A sequence number is a positive integer. A message is regarded as a duplicate and will be discarded if it has the same sequence number as one received before.

A digital signature S is computed by applying the sender's secret key SK and the decryption algorithm D on a time-stamp T, i.e.,

$$S = D(T,SK).$$

To verify this signature the receiver applies the sender's public key PK and the encryption algorithm E to the digital signature S, i.e.,

$$T = E(S,PK).$$

The digital signature is accepted if the time-stamp is valid (i.e., having the agreed time-keeper and correct sequence number). This confirms that the sender has signed the time-stamp since he is the only one who has the secret key to decrypt it.

The first four steps of the following time-stamped key distribution protocol for a videotex system are the same as

the first four steps of the process described above. Steps (5), (6) and (7) are replaced by the following:

(5)' User A chooses the prime numbers a, m, and a random positive integer x. The chip computes the value of $a^x \pmod m$, forms a digital signature S_a using a suitable time-stamp, and sends the following encrypted message to the IRC.

A-->IRC: $(a, m, a^x \pmod m, S_a, A)^{PK_{IRC}}$

This message is encrypted with the public key of IRC so that an intruder cannot intercept the digital signature S_a. The name of the sender, A, is also included so that the IRC knows A is the sender.

(6)' On receiving this message, the IRC decrypts it with his own secret key. It forms the communication key $a^{xy} \pmod m$ if A's digital signature is valid and the sequence number is not a duplicate. It computes the value $a^y \pmod m$ and forms a digital signature S_{IRC} using its own time-keeper and a new sequence number (e.g., the received sequence number + 1). The following encrypted message will then be sent to A:

IRC-->A: $(a^y \pmod m, S_{IRC}, IRC)^{PK_a}$

(7)' User A decrypts IRC's reply with his own secret key and verifies the received digital signature. After

checking the sequence number, he forms the communication key $a^{xy} \pmod{m}$.

Chapter III

CHI-SQUARE TEST FOR INDEPENDENCE OF ENCRYPTED DATA

3.1 INTRODUCTION

The following two problems are relevant to those schemes cryptanalysts often use for breaking a cryptosystem which uses a publicly available encryption algorithm:

Problem #1 For a fixed but arbitrary key, is there any relationship between a certain portion of the plaintext and a certain portion of the ciphertext?

Problem #2 For a fixed but arbitrary plaintext, is there any relationship between a certain portion of the key and a certain portion of the ciphertext?

If the answer to Problem #1 is 'yes', a cryptanalyst will be able to derive that portion of the plaintext by analyzing the corresponding portion of the ciphertext for a cryptosystem which is using a fixed key (though unknown to the attack). Although that portion may be just a small part of the entire plaintext, an experienced cryptanalyst may find it very helpful for conjecturing or deducing the whole

plaintext. Similarly, if the answer to Problem #2 is 'yes', a cryptanalyst will be able to derive the key by applying the algorithm on a known plaintext. In short, if some portions of the key, the plaintext and the ciphertext are found to be dependent, the encryption algorithm will become unsecure or even totally useless.

Thus, these two problems should be an important concern for encryption algorithm designers and users. However, as far as we know, very little research has been done with the objective of showing, either mathematically or experimentally, that the existing encryption algorithms are insensitive to this kind of troubles. Because of the complexity of these algorithms, rigorous mathematical proofs seem to be out of reach. In this Chapter, we investigate these two problems by means of a statistical technique, namely the Chi-square test for data independence. The technique is applicable to a general encryption algorithm. However, our tests are done on the DES (Data Encryption Standard) algorithm, because it is well known.

In Section 3.2, we review briefly the method of Chi-square test of independence. Section 3.3 describes our test process of applying this technique on the DES Algorithm. Section 3.4 presents the test results and discussions.

3.2 CHI-SQUARE TEST OF INDEPENDENCE

Suppose a sample of N outcomes have been collected from a population. These outcomes are grouped according to their values of two nominal variables describing the population. For example, a population of students may be described by their examination marks and their heights. We want to find out whether these two variables are independent or not by applying the Chi-square test on the observed frequencies of these groups.

Procedure of the Chi-square test

In a Chi-square test, we begin with the null hypothesis that the two nominal variables are independent. The test is then used to determine whether the hypothesis should be accepted or rejected. The process has the following steps:

i) Form a contingency table of cells using the two nominal variables as its row and column indices. The pair of values of the two variables for an outcome specify the cell it belongs to. After N outcomes are measured, the observed frequency O_{ij}, i.e., the number of outcomes belonging to cell (i,j), is recorded.

ii) From the observed values of this contingency table, calculate the expected frequencies E_{ij} as follows:

$$E_{ij} = \frac{R_i C_j}{N} \quad , \qquad \begin{matrix} i = 1,2,.....r. \\ j = 1,2,.....c. \end{matrix}$$

where E_{ij} = expected frequency for cell (i,j),

R_i = sum of the frequencies in row i,

C_j = sum of the frequencies in column j,

N = sum of the frequencies in all cells,

r = number of rows of the contingency table,

c = number of columns of the contingenct table.

iii) Compute the Chi-square value, χ^2 , by the following formula

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

where O_{ij}= observed frequency in cell (i,j) of the contingency table.

iv) Compare the computed Chi-square value χ^2 with a critical value obtained from a Chi-square table (also called rejection regions) at a specified significance level α and degree of freedom, df. Only a one-tailed test is appropriate.

v) If the computed Chi-square value equals or exceeds the Chi-square table value, the null hypothesis of independence is rejected. Otherwise, it is accepted.

Errors in hypothesis testing

When drawing conclusions on a hypothesis about a population by applying the Chi-square test on its samples, two types of errors may be committed:

Type I error: If the hypothesis is actually a true statement about the population but the test rejects it, a Type I error is committed. This type of error occurs with a probability α(alpha). That is, the probability of rejecting a true statement is no greater than α .

Type II error: If the hypothesis is actually a false statement about the population but the test accepts it, a Type II error is committed. This type of error occurs with a probability β(beta). That is, the probability of accepting a false statement is no greater than β .

The power of a statistical test on a hypothesis is defined as the probability that it will reject a false hypothesis. The value of the power is given as $(1-\beta)$. It is related to the sample size n, and the value α. Increase in n or α will decrease β and therefore will increase the power of a test (ROSC69).

3.3 APPLICATION OF CHI-SQUARE TEST ON THE DES ALGORITHM

We have conducted two sets of tests, one for Problem #1 and another for Problem #2. The test processes are explained below while the results will be described in Section 3.4.

i) Experimental process for Problem #1 (fixed key)

For Problem #1, with the key unchanged, the set of all possible (plaintext, ciphertext) pairs form the population. The two norminal variables describing the population are a specified portion of the plaintext and a specified portion of the ciphertext.

Each experiment consists of N cycles. Each cycle applies the DES algorithm once to get a (plaintext, ciphertext) pair as an outcome. The first cycle chooses an arbitary key and plaintext, both 64 bits long. In the other cycles, the same key is used but a random number generator is used to generate the plaintext.

In each cycle, the value of a specified portion of the plaintext and the value of a specified portion of the ciphertext are used as the row and column numbers, respectively, for determining the cell location in the contingency table. The frequency of that cell is then increased by one.

After N cycles, the Chi-square value is computed and the hypothesis is tested according to the method described in Section 3.2.

ii) Experimental process for Problem #2 (fixed plaintext)

The same concept and process as for Problem #1 are adopted for Problem #2, except that the plaintext and the key exchange their roles. The plaintext is now fixed in all cycles while, in each cycle, the ciphertext of the previous cycle is used as the new key.

3.4 TESTS AND DISCUSSIONS

Tests: (see also Section 3.3)

The data and results of our test are explained in the following:

i) Since software for the DES encryption and decryption algorithms is not available in the literature, a lot of efforts have been spent in their coding. Table 3.1 shows part of our tests on the validity of our coding. The 64-bit key and plaintext are chosen arbitrarily. The encryption algorithm is then applied to generate the 64-bit ciphertext. Then, this ciphertext is used as the plaintext for the decryption algorithm. The same key as for the encryption test is used. The output of the decryption test shows the complete recovery of the plaintext in the encryption test by the decryption algorithm.

ii) Twenty-one tests have been performed for each of Problem #1 and Problem #2. Results are summarised in Tables 3.2 and 3.3, respectively. In these tables, each row shows the data and the results of one test. The total frequency (i.e., sample size) for each test has been chosen between 500 to 3000, depending on the degree of freedom used. The bit positions of the plaintext and the ciphertext are chosen arbitrarily. Rejection regions are

obtained from the Table of Chi-square distribution [BEYE] with 1% significance level. The Chi-square values are calculated by the method described in Section 3.2.

For a statistical test, increase in the sample size will decrease the two types of errors mentioned in Section 3.2. Also, according to Roscoe (ROSC69), a sample size between 30 to 500 is usually adequate for this type of behavioural research. The sample size for our Chi-square data independence tests are between 500 to 3000. These are much greater than this requirement and should therefore give us a high level of confidence about our test results.

As shown in Tables 3.2 and 3.3, the computed Chi-square values are all less than the rejection regions at a significance level of 1%. Therefore, the hypothesis of independence between the specified portion of the plaintext and the specified portion of the ciphertext is accepted. The hypothesis of independence between the specified portion of the key and the specified portion of the ciphertext is also accepted in all of our tests.

iii) As illustrations, two typical examples of our Chi-square tests for Problem #1 and #2 are given and

shown in Figures 3.1 and 3.2. These figures are self-explanatory. The inputs and outputs for both examples are explained below and the Figure 3.3 shows the actual input data of Example 2.

The following data are input (see Figure 3.3):

* First line: Total frequency, Nnmber of bits to be tested in the plaintext or key, and number of bits to be tested in the ciphertext.
* Secone line: Positions of the bits in the plaintext/key.
* 3rd line: Positions of the bits in the ciphertext.
* 4th line: Fixed key? Print table? ('1' for 'yes', '0' for 'no').
* 5th line: significance level and rejection region.
* Lines 6 to 86: Data required for the DES algorithm (NBS 77).
* Last 4 lines: 64-bit long plaintext and key.

The following data are output (see Figures 3.1 and 3.2):

* Statement of hypothesis.
* The 64-bit fixed key (for Problem #1) or plaintext (for Problem #2) used.
* Contingency table (output is optional).

* Sum of the frequencies in all cells.
* Positions ofthe specidied but fixed portion of the plaintext/key.
* Positions of the specified but fixed portion of the ciphertext.
* Degrees of freedom.
* Significance level.
* Rejection regions.
* Computed Chi-square value.
* Conclusion about the hypothesis.
* Total CPU time for the run.

The results recorded in Figure 3.1 show that the hypothesis of independence between the specified portion (bits 1 and 2) of the plaintext and the specified portion (bits 3, 4 and 5) of the ciphertext is accepted. Also, the results recorded in Figure 3.2 show that the hypothesis of independence between the specified portion of the ciphertext (bits 1, 2 and 3) and the specified portion (bits 1, 2 and 3) of the key is accepted.

Table 3.1 Data of the DES Encryption and Decryption Algorithms.

Data for the DES encryption algorithm:

Key	10101110	00100110	10110000	00111100
	01001111	11110010	10011010	00001111
Input (Plaintext)	11001000	00111010	10111010	01111100
	10100101	10000001	11101010	01100011
Output (Ciphertext)	11100010	11000111	01101100	01101011
	00101100	10000101	01001101	00100011

Data of the DES decryption algorithm:

Key	10101110	00100110	10110000	00111100
	01001111	11110010	10011010	00001111
Input (Ciphertext)	11100010	11000111	01101100	01101011
	00101100	10000101	01001101	00100011
Output (Plaintext)	11001000	00111010	10111010	01111100
	10100101	10000001	11101010	01100011

Table 3.2 For a fixed key, Chi-square Independence Test between any portion of the plaintext and any portion of the ciphertext of the Data Encryption Standard Algorithm (DES).

Test No.	Total Frequency	Bit Positions of Plaintext	Bit Positions of Ciphertext	Degree of Freedom	Rejection Regions	Chi-square	Significance Level	Hypothesis
1	60	12, 8	23,37	9	21.67	5.15	0.01	accepted
2	60	9,13	57,64	9	21.67	4.83	0.01	accepted
3	60	59,44	63,55	9	21.67	13.22	0.01	accepted
4	60	43,58	63,42,59	21	38.93	29.75	0.01	accepted
5	60	14,23	42,57,64	21	38.93	16.75	0.01	accepted
6	60	8,24	56,43,61,64	45	69.96	45.42	0.01	accepted
7	60	20,30	27,32,19,40	45	69.96	50.38	0.01	accepted
8	60	7,17	21,34,47,62	45	69.96	35.93	0.01	accepted
9	60	3, 6	1,2,5,6	45	69.96	34.90	0.01	accepted
10	60	6,16,24	52,60	21	38.93	31.82	0.01	accepted
11	80	4,12,23	47,62	21	38.93	12.22	0.01	accepted
12	80	2, 9,10	15,16	21	38.93	29.14	0.01	accepted
13	60	3,10,21	41,57,62	49	74.92	46.98	0.01	accepted
14	60	20,35,40	56,59,63	49	74.92	55.95	0.01	accepted
15	60	40,55,59	48,63,60	49	74.92	36.17	0.01	accepted
16	70	12,23,51	5,14,32,60	105	141.62	95.29	0.01	accepted
17	80	1, 2, 3	1,2,3,4,5	217	271.12	245.99	0.01	accepted
18	60	43,59,61,50	60,54	45	69.96	38.97	0.01	accepted
19	60	55,49,62,63	39,48	45	69.96	47.10	0.01	accepted
20	60	2, 5,11,24	32,40	45	69.96	40.23	0.01	accepted
21	60	3,11,24,32	59,63	45	69.96	44.83	0.01	accepted

Table 3.3 For a fixed plaintext, Chi-square Independence Test between any portion of the key and any portion of the ciphertext of the Data Encryption Standard Algorithm (DES).

Test No.	Total Freqsency	Bit Positions of Key	Bit Positions of Ciphertext	Degree of Freedom	Rejection Regions	Chi-square	Significance Level	Hypothesis
1	50	1, 2	1, 2	9	21.67	13.03	0.01	accepted
2	50	1,12	23,58	9	21.67	7.92	0.01	accepted
3	50	1, 2	15,64	9	21.67	6.57	0.01	accepted
4	50	55,61	59,60	9	21.67	8.23	0.01	accepted
5	60	56,61	44,59,63	21	38.93	35.10	0.01	accepted
6	60	45,58	33,21,14	21	38.93	14.67	0.01	accepted
7	60	5, 8	47,52,59	21	38.93	32.32	0.01	accepted
8	60	21,33	46,52,58	21	38.93	23.63	0.01	accepted
9	60	62,45	63,54,42	21	38.93	30.63	0.01	accepted
10	60	5,11	15,23,32,39	45	69.96	69.17	0.01	accepted
11	60	23,34	44,56,63,64	45	69.96	49.52	0.01	accepted
12	60	45,57	42,49,59,62	45	69.96	45.85	0.01	accepted
13	50	6,18,24	52,60	21	38.93	24.06	0.01	accepted
14	80	55,63,60	42,35	21	38.93	18.80	0.01	accepted
15	50	23,55,63	1,15	21	38.93	21.89	0.01	accepted
16	70	3,4,5	3,4,5	49	74.92	55.33	0.01	accepted
17	70	1,9,18	23,45,61	49	74.92	57.54	0.01	accepted
18	70	1,2,3	1,2,3	49	74.92	55.70	0.01	accepted
19	60	62,45,2	63,54,42	49	74.92	55.00	0.01	accepted
20	70	12,23,51	5, 14,32,60	105	141.62	133.07	0.01	accepted
21	70	5,12,38,48	28,33,55,44	225	277.29	257.30	0.01	accepted

Example 1

```
HYPOTHESIS : FOR A FIXED KEY, A CHOSEN BUT FIXED PORTION OF CIPHERTEXT IS
             INDEPENDENT TO A CHOSEN BUT FIXED PORTION OF THE PLAINTEXT.

FIXED KEY : 10101110 00100110 10110000 00111100 01001111 11110010 10011010 00001111

CONTINGENCY TABLE :

       |  0  1  2  3  4  5  6  7   TOTAL
-------|-------------------------------
     0 | 13 17 15 27 17  9 10 14    122
     1 |  9 16 17 16 22 14 14 14    122
     2 | 20 15 22 16 19 14 15 13    134
     3 | 19 18 18 14 11 19 14  9    122
-------|-------------------------------
 TOTAL | 61 66 72 73 69 56 53 50    500

SUM OF THE FREQUENCIES FOR ALL CELLS  :  500
POSITIONS OF THE CHOSEN BUT FIXED PORTION OF THE PLAINTEXT:     1  2
POSITIONS OF THE CHOSEN BUT FIXED PORTION OF THE CIPHERTEXT:    3  4  5
DEGREE OF FREEDOM :            21
SIGNIFICANCE LEVEL:             0.0100
REJECTION REGIONS :            38.9300
CHI SQUARE :                   22.0154
CONCLUSION FROM THE TEST ABOUT THE HYPOTHESIS: THE HYPOTHESIS OF INDEPENDENCE IS ACCEPTED.
TOTAL CPU TIME USED :     21.2200 SECONDS.
```

Figure 3.1 Computer output of the Chi-square independence test between the plaintext and the ciphertext (for a fixed key) of the DES Algorithm.

Example 2

HYPOTHESIS : FOR A FIXED PLAINTEXT, A CHOSEN BUT FIXED PORTION OF THE CIPHERTEXT
IS INDEPENDENT TO A CHOSEN BUT FIXED PORTION OF THE KEY.

FIXED PLAINTEXT : 11001000 00111010 10111010 01111100 10100101 10000001 11101010 01100011

CONTINGENCY TABLE :

	0	1	2	3	4	5	6	7	TOTAL
0	10	9	11	15	6	7	5	6	69
1	6	8	12	5	5	11	7	9	64
2	10	7	8	9	6	3	8	6	57
3	12	7	10	7	4	13	12	7	72
4	8	8	4	5	7	10	7	4	53
5	6	8	5	9	6	10	11	10	65
6	10	7	3	14	10	5	9	9	67
7	7	10	4	8	9	5	8	5	56
TOTAL	69	62	57	72	53	64	67	56	500

SUM OF THE FREQUENCIES FOR ALL CELLS : 500

POSITIONS OF THE CHOSEN BUT FIXED PORTION OF THE KEY: 1 2 3

POSITIONS OF THE CHOSEN BUT FIXED PORTION OF THE CIPHERTEXT: 1 2 3

DEGREE OF FREEDOM : 49

SIGNIFICANCE LEVEL: 0.0100

REJECTION REGIONS : 74.9200

CHI SQUARE : 46.7069

CONCLUSION FROM THE TEST ABOUT THE HYPOTHESIS: THE HYPOTHESIS OF INDEPENDENCE IS ACCEPTED.

TOTAL CPU TIME USED : 21.2580 SECONDS.

Figure 3.2 Computer output of the Chi-square independence test between the key and the ciphertext (for a fixed plaintext) of the DES Algorithm.

```
500 3 3
1 2 3
1 2 3
0 1
0.01 74.92
58 50 42 34 26 18 10 2
60 52 44 36 28 20 12 4
62 54 46 38 30 22 14 6
64 56 48 40 32 24 16 8
57 49 41 33 25 17 9 1
59 51 43 35 27 19 11 3
61 53 45 37 29 21 13 5
63 55 47 39 31 23 15 7
40 8 48 16 56 24 64 32
39 7 47 15 55 23 63 31
38 6 46 14 54 22 62 30
37 5 45 13 53 21 61 29
36 4 44 12 52 20 60 28
35 3 43 11 51 19 59 27
34 2 42 10 50 18 58 26
33 1 41 9 49 17 57 25
32 1 2 3 4 5
4 5 6 7 8 9
8 9 10 11 12 13
12 13 14 15 16 17
16 17 18 19 20 21
20 21 22 23 24 25
24 25 26 27 28 29
28 29 30 31 32 1
14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7
0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8
4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0
15 12 8 2 4 9 1 7 5 11 3 14 10 0 6 13
15 1 8 14 6 11 3 4 9 7 2 13 12 0 5 10
3 13 4 7 15 2 8 14 12 0 1 10 6 9 11 5
0 14 7 11 10 4 13 1 5 8 12 6 9 3 2 15
13 8 10 1 3 15 4 2 11 6 7 12 0 5 14 9
10 0 9 14 6 3 15 5 1 13 12 7 11 4 2 8
13 7 0 9 3 4 6 10 2 8 5 14 12 11 15 1
13 6 4 9 8 15 3 0 11 1 2 12 5 10 14 7
1 10 13 0 6 9 8 7 4 15 14 3 11 5 2 12
7 13 14 3 0 6 9 10 1 2 8 5 11 12 4 15
13 8 11 5 6 15 0 3 4 7 2 12 1 10 14 9
10 6 9 0 12 11 7 13 15 1 3 14 5 2 8 4
3 15 0 6 10 1 13 8 9 4 5 11 12 7 2 14
2 12 4 1 7 10 11 6 8 5 3 15 13 0 14 9
14 11 2 12 4 7 13 1 5 0 15 10 3 9 8 6
4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14
11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3
12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11
10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8
9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6
4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13
4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1
```

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

Figure 3.3 Example of input data.

Discussions:

From the results of our tests, we can make the following remarks:

i) The DES algorithm is insensitive to cryptanalytic attacks

The results as recorded in Table 3.2 show that, at a 1% significance level, there is no close relationship between a portion of the plaintext and a portion of the ciphertext. Thus, a cryptananalyst will not be able to derive the plaintext from the ciphertext without knowing the encryption key. This implies that a DES cryptosystem can be ruled out from ciphertext-only attacks.

Similarly, the results as recorded in Table 3.3 show that, for a fixed plaintext, a portion of the key is independent of a portion of the ciphertext. Thus, it is impossible for a cryptanalyst to derive the key by applying the DES algorithm on a known plaintext. In other words, the known plaintext attack can also be ruled out for the DES algorithm.

ii) Applicability of our method to Telidon instructions

A Telidon system is more susceptible to ciphertext-only attacks because Telidon data streams have special structures. (See Section 1.3)

A Telidon PDI has only 32 different opcode formats. They are also publicly known. If the opcode portion of a PDI is found to be related with a certain portion of the ciphertext, a cryptanalyst will be able to deduce the most important part, i.e., the opcode, of the PDI without having to know the key being used. The probability of success in this kind of attacks is high since the number of posible PDI variations is small and they usually appear in certain patterns.

Our results imply that DES is a secure cryptosystem for data composed of Telidon instructions.

iii) Software implementation of encryption algorithm is impractical

As a side result, we can say something about the practicability of implementing DES by software. Our programs are written in PASCAL and run in an AMDAHL 470 V7-A computer.

On the average, the time required to encrypt or decrypt 64 bits of data by DES is about 52 milliseconds. Although this can probably be improved by optimizing the program coding, it is considered too slow for application to real-time information systems. For practical applications, hardware implementation is therefore necessary. A

microprocessor chip is now commercially available for the DES algorithm. It takes 25 microseconds to encrypt or decrypt eight bytes of data (DATA82). For the RSA algorithm, a CMOS LSI chip with encryption speed of 5000 bits/sec is also being designed (MIYA82).

Chapter IV

PROGRAM DESCRIPTION

4.1 DESCRIPTION OF PROGRAMS

This section outlines the computer programs of our experiments (see Chapter 3). Since coding of the DES algorithm is not available in the literature, great effort has been spent in coding this algorithm.

The software of our experiments consists of a main program, called CHI_SQUARE, and six procedures: INITIALIZATION, READ_DATA, DES, LOCATION, CHISQ and OUT_DATA, all coded in PASCAL. Figure 4.1 shows their logical relationship. They are described in more details in the following subsections.

4.1.1 Main program CHI-SQUARE

CHI_SQUARE is coded in such a way that it can perform several experiments in a single run. It defines the types and variables to be used in the accompanying subroutines and controls the interworking of the subroutines.

4.1.2 Procedure INITIALIZATION

This procedure inputs the values of the total frequency (i.e., number of experiments in a single run) for the chi square test, the number of plaintext and ciphertext bits to be tested and their positions, the significance level, and the rejection regions. Frequencies in all the cells of the contingency table are initialized to zero.

4.1.3 Procedure READ-DATA

This procedure inputs all the data (values for the permutation tables) required by the Data Encryption Standard (DES) Algorithm (NBS 77). The initial values of the key and plaintext are also input.

4.1.4 Procedure DES

This procedure implements the Data Encryption Standard (DES) Algorithm (NBS 77). It is supported by eight procedures: KEY_GENERATION, INITIAL_PERMUTATION, INVERSE_PERMUTATION, LR_SEPARATION, E_BIT_SELECTION, EXCLUSIVE_OR, PER_FUNCTION, and SELECTION_FUNCTION.

The algorithm's main logic is shown in Figure 4.2. Both its input (plaintext) and output (ciphertext) are 64-bit long. After permuting the plaintext, the 64-bit resultant block is divided into two equal halves. The right half undergoes a transposition, in which 16 of the 32 bits are duplicated to form a block of 48 bits. The input key is a

48-bit block and also undergoes some alterations. These two 48-bit blocks are then exclusive-ORed together, as shown in Figure 4.2. The result, when permuted, is 32 bits of data. Now that the right half has been transformed and exclusive-ORed to the appropriate key value. This is shown as f(R,K) in Figure 4.3. The result of that cipher is now exclusive-ORed with the left half from the initial permutation that has up to now been unchanged. That result will now be used as the right half for the next iteration of the entire process. As for the left half for the next iteration, the right half from the previous one will be used. This entire operation is repeated sixteen times, as shown in Figure 4.2.

4.1.5 Procedure KEY-GENERATION

This procedure is supported by three sub-procedures: PERMUTED_CHOICE_1, LEFT_SHIFT, and CHOICE_2. It generates 16 subkeys according to the method shown in Figure 4.4.

The initial 64-bit key is first reduced to 56 bits. It is then split into two equal halves, C and D. The two halves undergo a circular left shift or rotation, respectively. The number of left shifts is determined by the iteration number. The two halves are then recombined and undergo a permutation which reduces the number of bits to 48. It is used as the first subkey. The values of the rotated C and D become the new values of C and D of the next iternation. This subkey transformation is repeated sixteen times. The

permutation tables are the same for each calculation but the number of left shifts vary. For this reason, the sixteen iterations are not identical.

4.1.6 Procedure PERMUTED-CHOICE-1

This procedure supports the procedure KEY_GENERATION by transforming the original 64-bit key into a 56-bit permuted key which is then split into two equal halves, C and D.

4.1.7 Procedure LEFT-SHIFT

This procedure supports the procedure KEY_GENERATION by left shifting or rotating) the two halves C and D. The number of left shifts depends on the iteration number.

4.1.8 Procedure CHOICE-2

This procedure supports the procedure KEY_GENERATION by first recombining the two halves C and D into a 56-bit block and then reducing its number of bits to 48. This 48-bit block is then used as one of the 16 subkeys.

4.1.9 Procedure INITIAL-PERMUTATION

This procedure supports the procedure ENCRYPTION by permutating the original 64-bit input.

4.1.10 Procedure INVERSE-PERMUTATION

This procedure supports the procedure ENCRYPTION by permutating the 64-bit preoutput (see Figure 4.2) into ciphertext.

4.1.11 Procedure LR-SEPERATION

This procedure supports the procedure ENCRYPTION by dividing the 64-bit block into 2 equal parts, L and R.

4.1.12 Procedure E-BIT-SELECTION

This procedure supports the procedure ENCRYPTION by transposing and duplicating the right half R of 32 bits to produce a block of 48 bits.

4.1.13 Procedure EXCLUSIVE-OR

This procedure supports the procedure ENCRYPTION by performing bit-by-bit addition modulo 2 mathematics for a block of 32 or 48 bits.

4.1.14 Procedure PER-FUNCTION

This procedure supports the procedure ENCRYPTION by permuting the 32 bit block output from the procedure SELECTION_FUNCTION.

4.1.15 Procedure SELECTION-FUNCTION

This procedure supports the procedure ENCRYPTION by performing the following calculation: The 48-bit block output of the procedure EXCLUSIVE_OR is divided into 8 equal sub-blocks. Each block of 6 bits are input to SELECTION-FUNCTION, which yields a 4-bit output block according to the following procedures:

Assume B1 is a block of 6 bits and S1 is a defined selection table for B1. The first and last bits of B1 represent in base 2 a number in the range 0 to 3. Let that number be i. The middle 4 bits of B1 represent in base 2 a number in the range 0 to 15. Let that number be j. Look up in the table the number in the i'th row and j'th column. It is a number pre-defined by a 4-bit block. For example, for B1=011011 the row is 01 that is row 1, and the column is determined by 1101, that is column 13. If in row 1 column 13 appears 5, the output is 0101.

The procedure PRIMITIVE_FUNCTION is called to calculate the entry of the selection table and the suitable decimal to binary conversion.

4.1.16 Procedure PRIMITIVE-FUNCTION

This procedure supports the procedure SELECTION-FUNCTION by finding the entry of the table of the primitive function and performing suitable decimal-to-binary conversion.

4.1.17 Procedure LOCATION

This procedure supports the main program by determining the cell location in the contingency table. The value of a fixed portion or the plaintext and the value of a fixed portion of the ciphertext are used as the row and column number, respectively.

4.1.18 Procedure CHISQ

This procedure calculates the chi-square statistic by the method mentioned in Section 3.2.

4.1.19 Procedure OUT-DATA

This procedure outputs the relevant statistics. The contingency table is printed only on request by setting the variable PRINT_TABLE to 1.

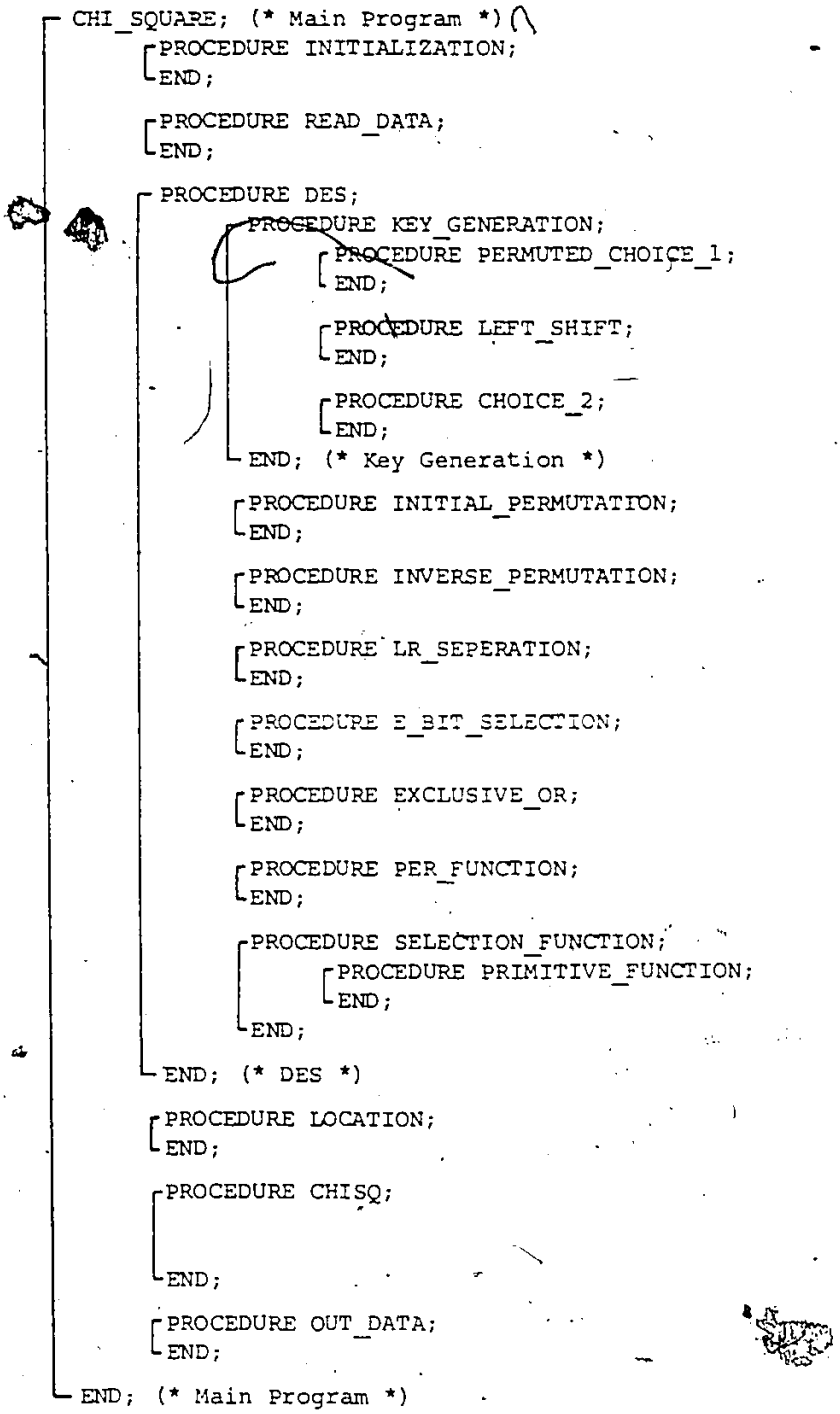


Figure 4.1 A schematic diagram of the computer program for Chi-square data independence tests.

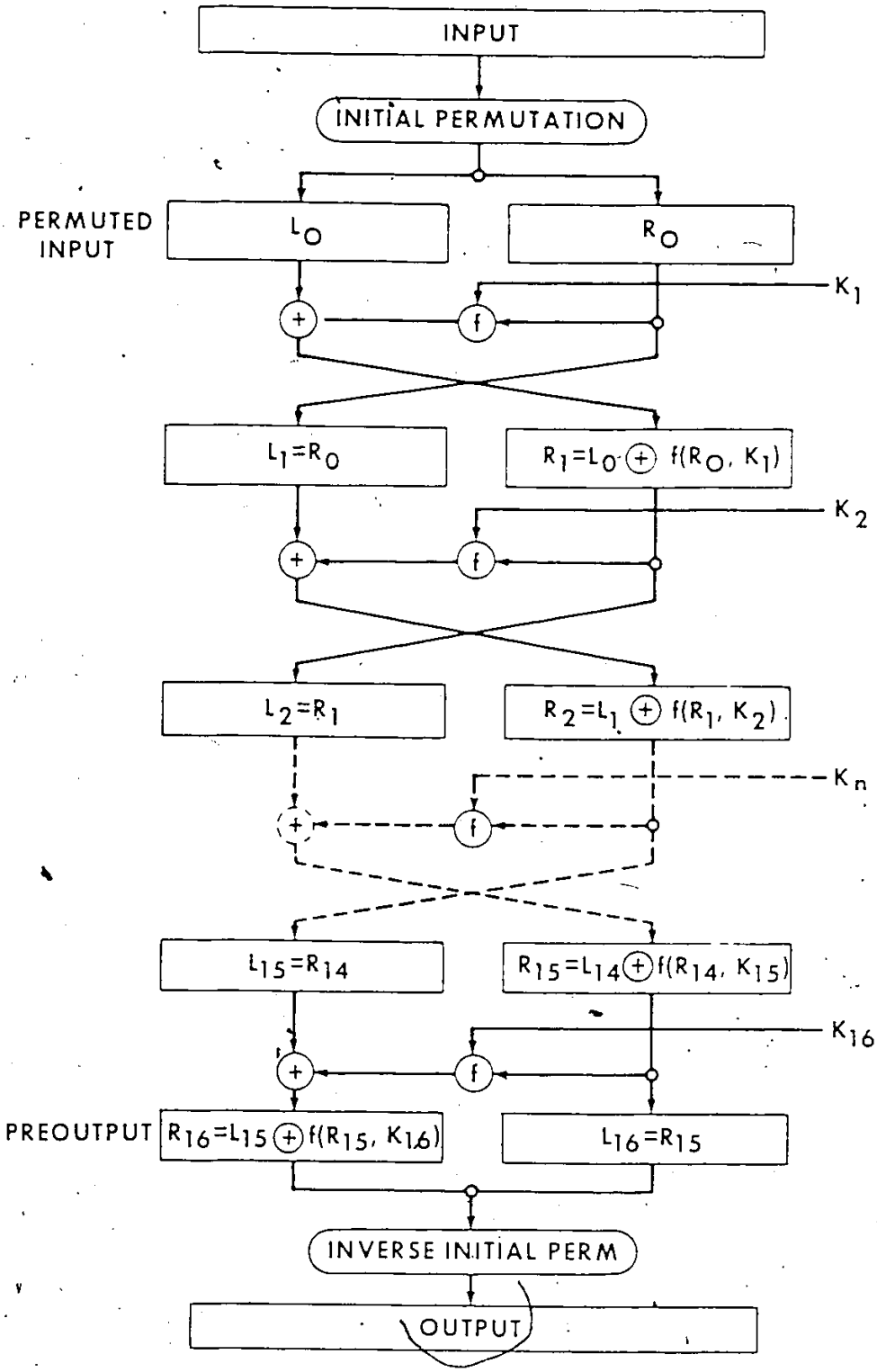


Figure 4.2 Flowchart for enciphering computation of the DES Algorithm.

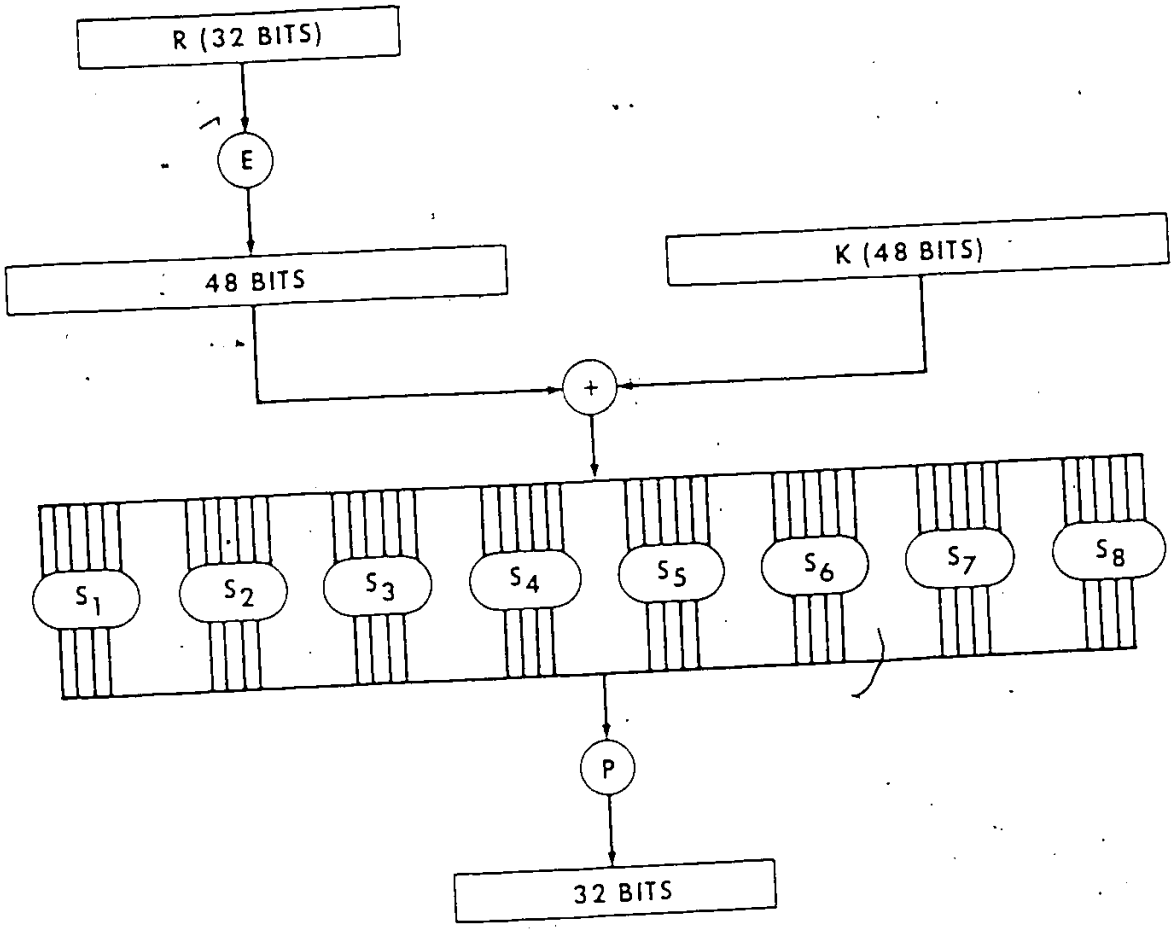


Figure 4.3 Flowchart for the calculation of f(R,K) in the DES Algorithm.

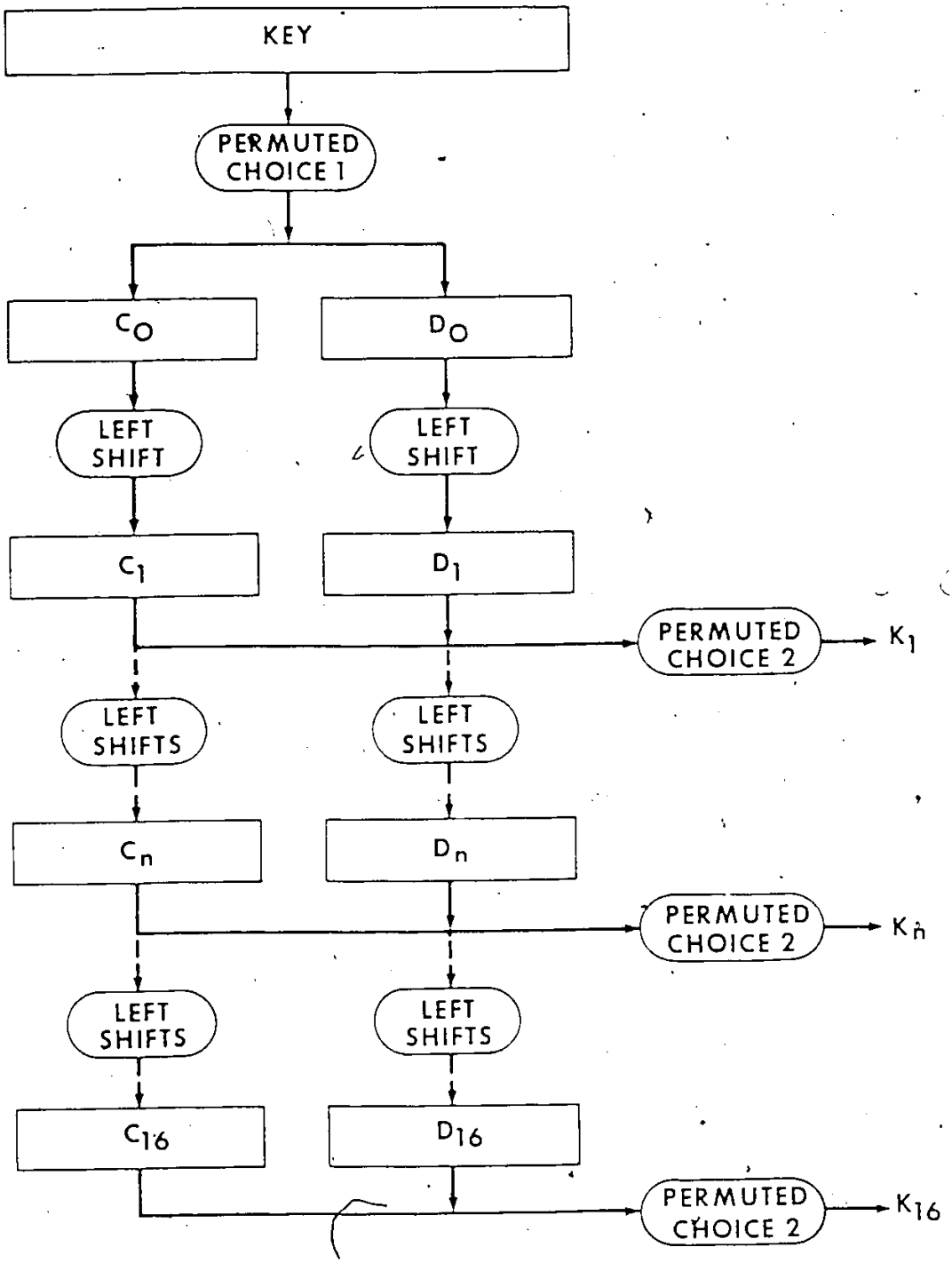


Figure 4.4 Flowchart for the key schedule calculation of the DES Algorithm.

4.2 PROGRAM FOR CHI-SQUARE TEST OF INDEPENDENCE

```
{*****************************************************************
*                                                    SEP 20,'82
*     MAIN PROGRAM FOR CHI SQUARE INDEPENDENCY TEST
*
*****************************************************************
*
*      THIS SYSTEM APPLIES THE STATISTCAL TECHNIQUE, NAMELY
*   THE CHI-SQUARE TEST FOR DATA INDEPENDENCE FOR SOLVING
*   THE FOLLOWING TWO PROBLEMS:
*
*      PROBLEM #1  FOR A FIXED BUT ARBITRARY KEY, IS THERE ANY
*                  RELATIONSHIP BETWEEN A CERTAIN PORTION OF
*                  THE PLAINTEXT AND A CERTAIN PORTION OF THE
*                  CIPHERTEXT?
*
*      PROBLEM #2  FOR A FIXED BUT ARBITRARY PLAINTEXT, IS
*                  THERE ANY RELATIONSHIP BETWEEN A CERTAIN
*                  PORTION OF THE KEY AND A CERTAIN PORTION OF
*                  THE CIPHERTEXT?
*
*      DATA IN BIT PATTERNS ARE ENCRYPTED BY THE DES (DATA
*   ENCRYPTION STANDARD) ALGORITHM OF NATIONAL BUREAUOF
*   STANDARDS OF USA.
*
*      THE SYSTEM CONSISTS OF A MAIN PROGRAM CHI-SQUARE AND
*   SIX PROCEDURES: INITIALIZATION, READ-DATA, DES, LOCATION,
*   CHISQ AND OUT-DATA, ALL CODED IN PASCAL.
*      THE MAIN PROGRAM IS DESIGNED TO PERFORM SEVERAL TESTS
*   IN A SINGLE RUN. IT FIRST DEFINES THE TYPE AND VARIABLES
*   TO BE USED IN THE ACCOMPANING SUBROUTINES.
*
*****************************************************************
*
*
* C:              VARIABLE USED FOR KEY SCHEDULE CALCULATION
*                 IN THE DES.
* CHI:            CONTINGENCY TABLE FOR CHI SQUARE TEST.
* CIPHERTEXT:     OUTPUT OF THE DATA ENCRYPTION STANDARD.
* COL_TOTAL:      COLUMN TOTAL IN THE CONTINGENCY TABLE.
* CONST_KEY:      FLAG TO BE SET IF THE KEY IS FIXED.
* COL_TEST_BIT:CONTAINS THE TESTING BIT POSITIONS IN
*                 THE PLAINTEXT.
* C_SQUARE:       CHI SQUARE.
* COL:            COLUMN LOCATION IN THE CONTINGENCY TABLE.
* CPU_1:          CONTAINS THE TIME TO START EXECUE THE PROGRAM.
* CPU_2:          CONTAINS THE FINAL TIME TO EXECUE THE PROGRAM.
* D:              VARIABLE USED FOR KEY SCHOULE CALCULATION IN
*                 THE DES.
* DEGREE_OF_FREEDOM: DEGREE OF FREEDOM.
* E:              E BIT-SELECTION TABLE IN THE DES.
* EXPECT:         EXPECTATION IN THE CONTINGENCY TABLE.
* FRK:            VALUES FOR F(R,K) IN THE DES.
* IP1:            INITIAL PERMUTATION TABLE IN THE DES.
* IP2:            INVERSE OF THE INITIAL PERMUTATION TABLE IN
*                 THE DES.
* INPUT_TEST_BIT: NO. OF BITS TO BE TESTED IN THE PLAINTEXT (INPUT).
* INPUT_SCALE:    THE TOTAL NO. OF ROWS OCCUPYED IN THE
*                 CONTINGENCY TABLE IN THE PRESENT TESTING.
* KEY:            KEY OF THE DES.
* K1,K2,....K16:16 SUBKEYS GENERATED FROM THE KEY SCHEDULE
*                 CALCULATION IN DES.
```

```
* L:              VARIABLE FOR ENCIPHERING COMPUTATION IN THE
*                  DES.
* LSHIFT:          NO. OF LEFT SHIFTS WITHIN THE KEY SCHEDULE IN
*                  THE DES.
* NO_OF_TEST:      TOTAL NO. OF FREQUENCY FOR THE CHI SQUARE TEST.
* OUTPUT_TEST_BIT:NO. OF BITS TO BE TESTED IN THE CIPHERTEXT.
* OUTPUT_SCALE:THE TOTAL NO. OF COLUMNS OCCUPYED IN THE
*                  CONTINGENCY TABLE IN THE PRESENT TESTING.
* P:               PERMUTATION FUNCTION P USED IN THE DES.
* PC1:             PERMUTED CHOICE 1 IN THE DES.
* PC2              PERMUTED CHOICE 2 IN THE DES.
* PLAINTEXT        64 BITS INPUT FOR THE DES.
* PRINT_TABLE:     FLAG TO INDICATE WHETHER THE CONTINGENCY
*                  TABLE IS GOING TO BE PRINTED.
* R:               VARIABLE FOR ENCIPHER COMPUTATION IN THE DES.
* RR:              OUTPUT AFTER APPLYING THE E-BIT-SELECTION
*                  FUNCTION TO THE BITS IN R.
* RRK:             VALUE OBTAINED FROM EXCLUSIVE OR OF THE 48
*                  BITSUBKEY AND RR.
* ROW:             ROW LOCATION IN THE CONTINGENCY TABLE.
* ROW_TEST_BIT:CONTAINS BIT POSITIONS IN THE PLAINTEXT FOR THE
*                  CHI SQUARE TEST.
* ROW_TOTAL:       ROW TOTAL IN THE CONTINGENCY TABLE.
* S1,S2,....,S8:8 PRIMITIVE FUNCTIONS FOR THE DES.
* S:               32 BIT OUTPUT FROM THE 8 PRIMITIVE FUNCTION
*                  IN THE CALCULATION OF F(R,K) IN THE DES.
* TABLE_VALUE: CONFIDENCE LEVEL.
*
*****************************************************************
*)

    PROGRAM CHI_SQUARE(INPUT,OUTPUT);
    TYPE
      MAT_8 = ARRAY[.1..8.] OF INTEGER;
      MAT_256_256 = ARRAY[.0..255, 0..255.] OF INTEGER;
      MAT_64 = ARRAY[.1..64.] OF INTEGER;
      MAT_256 = ARRAY[.0..255.] OF INTEGER;
      MAT_255_255 = ARRAY[.0..255, 0..255.] OF REAL;
      MAT_32 = ARRAY[.1..32.] OF INTEGER;
      MAT_28 = ARRAY[.1..28.] OF INTEGER;
      MAT_16 = ARRAY[.1..26.] OF INTEGER;
      MAT_4  = ARRAY[.1..4.]  OF INTEGER;
      MAT_48 = ARRAY[.1..48.] OF INTEGER;
      MAT_56 = ARRAY[.1..56.] OF INTEGER;
      MAT_4_16 = ARRAY[.0..3, 0..15.] OF INTEGER;
    VAR
      IP1,IP2,KEY : MAT_64;
      E,RR,RRK,PC2 : MAT_48;
      S1,S2,S3,S4,S5,S6,S7,S8 : MAT_4_16;
      P,L,R,S,FRK,STORE_L,STORE_R : MAT_32;
      PC1 : MAT_56;
      LSHIFT : MAT_16;
      C,D : MAT_28;
      K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,K11,K12,K13,K14,K15,K16:MAT_48;
      ROW_TEST_BIT,COL_TEST_BIT : MAT_8;
      CHI : MAT_256_256;
      EXPECT : MAT_255_255;
      CIPHERTEXT,PLAINTEXT : MAT_64;
      COL_TOTAL,ROW_TOTAL : MAT_256;
      ROW,COL,K,J,I,NO_OF_TEST,DEGREE_OF_FREEDOM : INTEGER;
      INPUT_TEST_BIT,OUTPUT_TEST_BIT,INPUT_SCALE,OUTPUT_SCALE:INTEGER;
      SIGNIFICANCE_LEVEL,C_SQUARE,TABLE_VALUE :REAL;
      PRINT_TABLE,CONST_KEY,CPU_1,CPU_2 : INTEGER;
```

```
{*************************************************************}

      PROCEDURE INITIALIZATION;

{*************************************************************
 *
 *      THIS PROCEDURE INPUTS THE VALUES OF THE TOTAL FREQUENCY
 *   OF THE CHI-SQUARE TEST, THE NUMBER OF PLAINTEXT AND
 *   CIPHERTEXT BITS TO BE TESTED AND THEIR POSITIONS, THE
 *   SIGNIFICANCE LEVEL, AND THE REJECTION REGIONS. ALL THE
 *   VARIABLES IN THE CONTINGENCY TABLE ARE INITIALIZED TO BE
 *   ZERO.
 *
 *************************************************************}

      VAR
        I,J,K : INTEGER;
      BEGIN
        READ(NO_OF_TEST, INPUT_TEST_BIT, OUTPUT_TEST_BIT);
        READLN;
        FOR K:= 1 TO INPUT_TEST_BIT DO
            READ(ROW_TEST_BIT(.K.));
        READLN;

        FOR K:= 1 TO OUTPUT_TEST_BIT DO
            READ(COL_TEST_BIT(.K.));
        READLN;

        READ (CONST_KEY, PRINT_TABLE);
        READLN;

        READ (SIGNIFICANCE_LEVEL, TABLE_VALUE);
        READLN;

        INPUT_SCALE:=2**INPUT_TEST_BIT -1;
        OUTPUT_SCALE:=2**OUTPUT_TEST_BIT -1;
        FOR I:= 0 TO INPUT_SCALE DO
            FOR J:= 0 TO OUTPUT_SCALE DO
                CHI(.I,J.) := 0;

            FOR K:= 0 TO INPUT_SCALE DO
                ROW_TOTAL(.K.) := 0;

            FOR K:= 0 TO OUTPUT_SCALE DO
                COL_TOTAL(.K.) := 0;

            C_SQUARE := 0;
      END; (* INITIALIZATION *)


{*************************************************************}

      PROCEDURE READ_DATA;

{*************************************************************
 *
 *      THIS PROCEDURE INPUTS ALL THE DATA REQUIRED BY THE DES
 *   (DATA ENCRYPTION STANDARD) ALGORITHM. INITIAL VALUES OF
 *   THE KEY AND PLAINTEXT ARE ALSO INPUT.
 *
 *************************************************************}

      VAR
        I,J : INTEGER;
      BEGIN
        FOR I:= 1 TO 64 DO
            READ (IP1(.I.));
        READLN;

        FOR I:= 1 TO 64 DO
            READ (IP2(.I.));
        READLN;

        FOR I:= 1 TO 48 DO
            READ (E(.I.));
        READLN;
```

```
FOR I:= 0 TO 3 DO
    FOR J:= 0 TO 15 DO
        READ (S1(.I,J.));
READLN;

FOR I:= 0 TO 3 DO
    FOR J:= 0 TO 15 DO
        READ (S2(.I,J.));
READLN;

FOR I:= 0 TO 3 DO
    FOR J:= 0 TO 15 DO
        READ (S3(.I,J.));
READLN;

FOR I:= 0 TO 3 DO
    FOR J:= 0 TO 15 DO
        READ (S4(.I,J.));
READLN;

FOR I:= 0 TO 3 DO
    FOR J:= 0 TO 15 DO
        READ (S5(.I,J.));
READLN;

FOR I:= 0 TO 3 DO
    FOR J:= 0 TO 15 DO
        READ (S6(.I,J.));
READLN;

FOR I:= 0 TO 3 DO
    FOR J:= 0 TO 15 DO
        READ (S7(.I,J.));
READLN;

FOR I:= 0 TO 3 DO
    FOR J:= 0 TO 15 DO
        READ (S8(.I,J.));
READLN;

FOR I:= 1 TO 32 DO
    READ (P(.I.));
READLN;

FOR I:= 1 TO 56 DO
    READ (PC1(.I.));
READLN;

FOR I:= 1 TO 48 DO
    READ (PC2(.I.));
READLN;

FOR I:= 1 TO 16 DO
    READ (LSHIFT(.I.));
READLN;

FOR I:= 1 TO 64 DO
    READ (CIPHERTEXT(.I.));
READLN;


FOR I:= 1 TO 64 DO
    READ (KEY(.I.));

END; (* READ_DATA *)
```

```
{*****************************************************************}

      PROCEDURE DES(INDICATOR:INTEGER);

{*****************************************************************
 *
 *      THIS PROCEDURE IMPLEMENTS THE DATA ENCRYPTION STANDARD
 *   ALGORITHM. IT IS SUPPORTED BY EIGHT PROCEDURES:
 *   KEY-GENERATION, INITIAL-PERMUTATION, INVERSE-PERMUTATION
 *   LR-SEPARATION, E-BIT-SELECTION, EXCLUSIVE-OR, PER-FUNCTION,
 *   AND SELECTION-FUNCTION.
 *   INDICATOR: THE VARIABLE INDICATING THE CYCLE NUMBER.
 *
 *****************************************************************}

      VAR
        STEP,A,B : INTEGER;


{*****************************************************************}

      PROCEDURE KEY_GENERATION;

{*****************************************************************
 *
 *      THIS PROCEDURE IS SUPPORTED BY THREE PROCEDURES TO
 *   GENERATE 16 SUBKEYS: PERMUTED_CHOICE_1, LEFT_SHIFT, AND
 *   CHOICE_2.
 *
 *****************************************************************}

      VAR
        M : INTEGER;

{*****************************************************************}

      PROCEDURE PERMUTED_CHOICE_1;

{*****************************************************************
 *
 *      THIS PROCEDURE SUPPORTS THE PROCEDURE KEY-GENERATION
 *   BY TRANSFORMING THE ORIGINAL 64-BIT KEY INTO A 56-BIT
 *   PERMUTED KEY, WHICH IS THEN SPLIT INTO TWO EQUAL HALVES,
 *   C AND D.
 *
 *****************************************************************}

      VAR
        I,J,INDEX : INTEGER;
      BEGIN
        FOR I:= 1 TO 28 DO
            BEGIN
              INDEX:= PC1(.I.);
              C(.I.) := KEY(.INDEX.);
            END; (* FOR *)

        FOR J:= 1 TO 28 DO
            BEGIN
              INDEX := PC1(.J+28.);
              D(.J.) := KEY(.INDEX.);
            END; (* FOR *)
      END; (* PERMUTED_CHOICE_1 *)
```

```
{*****************************************************************}

       PROCEDURE LEFT_SHIFT(NUMBER:INTEGER);

{*****************************************************************
 *
 *       THIS PROCEDURE SUPPORTS THE PROCEDURE KEY-GENERATION BY
 *    LEFT SHIFTING (OR ROTATING) THE TWO HALVES C AND D. THE
 *    NUMBER OF LEFT SHIFTS IS DETERMINED BY THE ITERATION
 *    NUMBER.
 *
 *****************************************************************}

      VAR
        SHIFT_NUM,I,TEMP1,TEMP2,J : INTEGER;
      BEGIN
        SHIFT_NUM := LSHIFT(.NUMBER.);
        FOR I:= 1 TO SHIFT_NUM DO
            BEGIN
              TEMP1 := C(.1.);
              TEMP2 := D(.1.);
              FOR J:= 1 TO 27 DO
                  BEGIN
                    C(.J.) := C(.J+1.);
                    D(.J.) := D(.J+1.);
                  END; (* FOR *)
              C(.28.) := TEMP1;
              D(.28.) := TEMP2;
            END; (* FOR *)
      END; (* LEFT_SHIFT *)
```

```
{*****************************************************************}

      PROCEDURE CHOICE_2(LOC : INTEGER);

{*****************************************************************
 *
 *      THIS PROCEDURE SUPPORTS THE PROCEDURE KEY-GENERATION BY
 *   FIRST RECOMBINING THE TWO HALVES C AND D INTO A 56-BIT
 *   BLOCK AND THEN REDUCING ITS NUMBER OF BITS TO 48. THIS
 *   48-BIT BLOCK IS USED AS ONE OF THE 16 SUBKEYS.
 *
 *****************************************************************}

      VAR
        CD : MAT_56;
        I,J,K,INDEX : INTEGER;
      BEGIN
        FOR I:= 1 TO 28 DO
            CD[.I.] := C[.I.];

        FOR J:= 1 TO 28 DO
            CD[.J+28.] := D[.J.];

        FOR K:= 1 TO 48 DO
            BEGIN
              INDEX := PC2[.K.];
              CASE LOC OF
                1: K1[.K.] := CD[.INDEX.];
                2: K2[.K.] := CD[.INDEX.];
                3: K3[.K.] := CD[.INDEX.];
                4: K4[.K.] := CD[.INDEX.];
                5: K5[.K.] := CD[.INDEX.];
                6: K6[.K.] := CD[.INDEX.];
                7: K7[.K.] := CD[.INDEX.];
                8: K8[.K.] := CD[.INDEX.];
                9: K9[.K.] := CD[.INDEX.];
               10: K10[.K.]:= CD[.INDEX.];
               11: K11[.K.]:= CD[.INDEX.];
               12: K12[.K.]:= CD[.INDEX.];
               13: K13[.K.]:= CD[.INDEX.];
               14: K14[.K.]:= CD[.INDEX.];
               15: K15[.K.]:= CD[.INDEX.];
               16: K16[.K.]:= CD[.INDEX.];
              END; (* CASE *)
            END; (* FOR *)
      END; (* PERMUTED_CHOICE_2 *)

      BEGIN    (* KEY_GENERATION *)
        PERMUTED_CHOICE_1;
        FOR M := 1 TO 16 DO
            BEGIN
              LEFT_SHIFT(M);
              CHOICE_2(M);
            END; (* FOR *)
      END; (* KEY_GENERATION *)
```

```
{*************************************************************}

      PROCEDURE INITIAL_PERMUTATION;

{*************************************************************
 *
 *      THIS PROCEDURE SUPPORTS THE PROCEDURE ENCRYPTION BY
 *   PERMUTATING THE ORIGINAL 64 BITS INPUT.
 *
 *************************************************************}

      VAR
        TEMP : MAT_64;
        I,INDEX : INTEGER;
      BEGIN
        FOR I:= 1 TO 64 DO
            BEGIN
              INDEX := IP1(.I.);
              TEMP(.I.) := CIPHERTEXT(.INDEX.);
            END; (* FOR *)
        CIPHERTEXT := TEMP;
      END; (* INITIAL_PERMUTATION *)



{*************************************************************}

      PROCEDURE INVERSE_PERMUTATION;

{*************************************************************
 *
 *      THIS PROCEDURE SUPPORTS THE PROCEDURE ENCRYPTION BY
 *   PERMUTATING THE PREOUTPUT INTO CIPHERTEXT.
 *
 *************************************************************}

      VAR
        TEMP : MAT_64;
        I,INDEX : INTEGER;
      BEGIN
        FOR I:= 1 TO 64 DO
            BEGIN
              INDEX := IP2(.I.);
              TEMP(.I.) := CIPHERTEXT(.INDEX.);
            END; (* FOR *)
        CIPHERTEXT  := TEMP;
      END; (* INVERSE_PERMUTATION *)


{*************************************************************}

      PROCEDURE LR_SEPERATION;

{*************************************************************
 *
 *      THIS PROCEDURE SUPPORTS THE PROCEDURE ENCRYPTION BY
 *   DIVIDING THE 64-BIT BLOCK INTO TWO EQUAL PARTS L AND R.
 *
 *************************************************************}

      VAR
        I,J : INTEGER;
      BEGIN
        FOR I:= 1 TO 32 DO
            L(.I.) := CIPHERTEXT(.I.);

        FOR J:= 1 TO 32 DO
            R(.J.) := CIPHERTEXT(.J+32.);
      END;  (* LR_SEPERATION *)
```

```
{*****************************************************************}

      PROCEDURE E_BIT_SELECTION;

{*****************************************************************
 *
 *      THIS PROCEDURE SUPPORTS THE PROCEDURE ENCRYPTION BY
 *   TRANSPOSING AND DUPLICATING THE RIGHT HALF R OF 32 BITS
 *   TO PRODUCE A BLOCK OF 48 BITS.
 *
 *****************************************************************}

      VAR
        I,INDEX : INTEGER;

      BEGIN
        FOR I:=1 TO 48 DO
            BEGIN
              INDEX := E[.I.];
              RR[.I.] := STORE_R[.INDEX.];
            END; (* FOR *)
      END; (* E_BIT_SELECTION *)

{*****************************************************************}

      PROCEDURE EXCLUSIVE_OR(DIMENSION,NUM : INTEGER);

{*****************************************************************
 *
 *      THIS PROCEDURE SUPPORTS THE PROCEDURE ENCRYPTION BY
 *   PERFORMING BIT-BY-BIT ADDITION MODULO 2 MATHEMATICS ON A
 *   A BLOCK OF 32 OR 48 BITS.
 *
 *****************************************************************}

      VAR
        I : INTEGER;
      BEGIN
        CASE DIMENSION OF
          32: FOR I:= 1 TO 32 DO
              R[.I.]:=(STORE_L[.I.]+FRK[.I.]) MOD 2;

          48: FOR I:= 1 TO 48 DO
              CASE NUM OF
                1: RRK[.I.]:=(RR[.I.]+K1[.I.]) MOD 2;
                2: RRK[.I.]:=(RR[.I.]+K2[.I.]) MOD 2;
                3: RRK[.I.]:=(RR[.I.]+K3[.I.]) MOD 2;
                4: RRK[.I.]:=(RR[.I.]+K4[.I.]) MOD 2;
                5: RRK[.I.]:=(RR[.I.]+K5[.I.]) MOD 2;
                6: RRK[.I.]:=(RR[.I.]+K6[.I.]) MOD 2;
                7: RRK[.I.]:=(RR[.I.]+K7[.I.]) MOD 2;
                8: RRK[.I.]:=(RR[.I.]+K8[.I.]) MOD 2;
                9: RRK[.I.]:=(RR[.I.]+K9[.I.]) MOD 2;
               10: RRK[.I.]:=(RR[.I.]+K10[.I.]) MOD 2;
               11: RRK[.I.]:=(RR[.I.]+K11[.I.]) MOD 2;
               12: RRK[.I.]:=(RR[.I.]+K12[.I.]) MOD 2;
               13: RRK[.I.]:=(RR[.I.]+K13[.I.]) MOD 2;
               14: RRK[.I.]:=(RR[.I.]+K14[.I.]) MOD 2;
               15: RRK[.I.]:=(RR[.I.]+K15[.I.]) MOD 2;
               16: RRK[.I.]:=(RR[.I.]+K16[.I.]) MOD 2;
              END; (* CASE *)
            END; (* CASE *)
      END; (* EXCLUSIVE_OR *)
```

```
{*****************************************************************}

      PROCEDURE PER_FUNCTION;

{*****************************************************************
 *
 *     THIS PROCEDURE SUPPORTS THE PROCEDURE ENCRYPTION BY
 *   PERMUTING THE 32 BIT BLOCK OUTPUT FROM THE PROCEDURE
 *   SELECTION_FUNCTION.
 *
 *****************************************************************}

      VAR
        I,INDEX : INTEGER;
      BEGIN
        FOR I:= 1 TO 32 DO
            BEGIN
              INDEX := P(.I.);
              FRK(.I.) := S(.INDEX.);
            END; (* FOR *)
      END; (* PER_FUNCTION *)


{*****************************************************************}

      PROCEDURE SELECTION_FUNCTION;

{*****************************************************************
 *
 *     THIS PROCEDURE SUPPORTS THE PROCEDURE ENCRYPTION BY
 *   PERFORMING THE FOLLOWING CALCULATION: THE 48-BIT BLOCK
 *   OUTPUT FROM THE PROCEDURE EXCLUSIVE_OR IS DIVIDED INTO
 *   8 SUB-BLOCKS. EACH BLOCK TAKES 6 BITS AS INPUT AND YIELDS
 *   4 BITS AS OUTPUT.
 *
 *****************************************************************}

      VAR
        BB : MAT_4;
        INDEX,J,K : INTEGER;


{*****************************************************************}

      PROCEDURE PRIMITIVE_FUNCTION(POS:INTEGER;SN:MAT_4_16);

{*****************************************************************
 *
 *     THIS PROCEDURE SUPPORTS THE PROCEDURE SELECTION-
 *   FUNCTION BY FINDING THE ENTRY OF THE SELECTION TABLE AND
 *   PERFORMING SUITABLE DECIMAL TO BINARY CONVERSION.
 *
 *****************************************************************}

      VAR
        ROW,COLUMN,NUM,I : INTEGER;
        VALUE1, VALUE2,VALUE3 : INTEGER;
      BEGIN
        VALUE1:=RRK(.POS+5.)*2 + RRK(.POS.);
        CASE VALUE1 OF
          0: ROW := 0;
          1: ROW := 1;
          2: ROW := 2;
          3: ROW := 3;
        END; (* CASE *)
```

```
    VALUE2:=RRK[.POS+4.]*8 + RRK[.POS+3.]*4
            + RRK[.POS+2.]*2 + RRK[.POS+1.];
    CASE VALUE2 OF
      0: COLUMN := 0;
      1: COLUMN := 1;
      2: COLUMN := 2;
      3: COLUMN := 3;
      4: COLUMN := 4;
      5: COLUMN := 5;
      6: COLUMN := 6;
      7: COLUMN := 7;
      8: COLUMN := 8;
      9: COLUMN := 9;
     10: COLUMN :=10;
     11: COLUMN :=11;
     12: COLUMN :=12;
     13: COLUMN :=13;
     14: COLUMN :=14;
     15: COLUMN :=15;
    END; (* CASE *)

    NUM := SN[.ROW, COLUMN.];
    FOR I:= 4 DOWNTO 1 DO
        BEGIN
          BB[.I.] := NUM MOD 2;
          NUM := NUM DIV 2;
        END; (* FOR *)
  END; (* PRIMITIVE_FUNCTION *)

  BEGIN  (* SELECTION_FUNCTION *)
    INDEX := 1;
    FOR J:= 1 TO 8 DO
        BEGIN
          CASE J OF
            1: PRIMITIVE_FUNCTION(1,S1);
            2: PRIMITIVE_FUNCTION(7,S2);
            3: PRIMITIVE_FUNCTION(13,S3);
            4: PRIMITIVE_FUNCTION(19,S4);
            5: PRIMITIVE_FUNCTION(25,S5);
            6: PRIMITIVE_FUNCTION(31,S6);
            7: PRIMITIVE_FUNCTION(37,S7);
            8: PRIMITIVE_FUNCTION(43,S8);
          END; (* CASE *)

    FOR K:= 1 TO 4 DO
        BEGIN
          S[.INDEX.] := BB[.K.];
          INDEX := INDEX + 1;
        END; (* FOR *)
    END; (* FOR *)
  END; (* SELECTION_FUNCTION *)


(*  *** IF INDICATOR = 1, (FIRST CYCLE), PLAINTEXT IS OBTAINED***
    *** FROM INPUT. OTHERWISE, PLAINTEXT OR KEY IS OBTAINED   ***
    *** FROM THE PREVIOUS CIPHERTEXT.                         ***
*)
  BEGIN (* DES *)
    IF INDICATOR = 1
    THEN  PLAINTEXT := CIPHERTEXT
    ELSE  IF CONST_KEY = 1
          THEN  PLAINTEXT := CIPHERTEXT
          ELSE  KEY := CIPHERTEXT;

    KEY_GENERATION;
    INITIAL_PERMUTATION;
    LR_SEPERATION;
```

```
{*      *** 16 ITERATIONS ARE GOING TO PERFORM WITH THE PRESENT  ***
        *** OF THE SUITABLE SUBKEY.                              ***
*}
          FOR STEP := 1 TO 16 DO
              BEGIN
                STORE_L := L;
                L := R;
                STORE_R:=R;
                E_BIT_SELECTION;
                EXCLUSIVE_OR(48,STEP);
                SELECTION_FUNCTION;
                PER_FUNCTION;
                EXCLUSIVE_OR(32,STEP);
              END; (* FOR *)

{*      *** THE RESULT OF THE 16 TH ITERATION IS ASSIGNED TO BE  ***
        *** THE PREOUTPUT.                                       ***
*}
          FOR A:= 1 TO 32 DO
              CIPHERTEXT(.A.) := R(.A.);
          FOR B:= 1 TO 32 DO
              CIPHERTEXT(.B+32.) := L(.B.);
          INVERSE_PERMUTATION;

        END; (* DES *)




{************************************************************

         PROCEDURE  LOCATION (TEMP : MAT_64;VAR PLACE:INTEGER;
                                                FLAG :INTEGER);

{************************************************************
 *
 *       THIS PROCEDURE SUPPORTS THE MAIN PROGRAM BY DETERMINING
 *    THE LOCATION OF THE CELL IN THE CONTINGENCY TABLE.
 *    FLAG:  INDICATING ROW OR COLUMN LOCATION IN USED.
 *    PLACE: VARIABLE WHCIH CONTAINS THE ROW OR COLUMN LOCATION.
 *
 ************************************************************

        VAR
          I,INDEX : INTEGER;
        BEGIN
          PLACE := 0;
          CASE FLAG OF
            0: FOR I:= 1 TO INPUT_TEST_BIT DO
                    BEGIN
                      INDEX:=ROW_TEST_BIT(.I.);
                      PLACE:=PLACE+TEMP(.INDEX.)*2**(I-1);
                    END;

            1: FOR I:= 1 TO OUTPUT_TEST_BIT DO
                    BEGIN
                      INDEX:=COL_TEST_BIT(.I.);
                      PLACE:=PLACE+TEMP(.INDEX.)*2**(I-1);
                    END;
          END; (* CASE *)
        END; (* LOCATION *)
```

```
{*****************************************************************
*}
      PROCEDURE CHISQ;
{*
*****************************************************************
*
*     THIS PROCEDURE SUPPORTS THE MAIN PROGRAM BY CALCULATING
*  THE CHI-SQUARE STATISTIC VALUE.
*
*****************************************************************
*}
      VAR
        I,J :INTEGER;

      BEGIN (* CHISQ *)

{*    *** THE FOLLOWING LINES CALCULATE THE ROW TOTAL, COLUMN ***
      *** TOTAL, EXPECTATION AND THE CHI-SQUARE VALUE, AND    ***
      *** THE DEGREE OF FREEDOM.                              ***
-*}

        FOR I:= 0 TO INPUT_SCALE DO
            FOR J:= 0 TO OUTPUT_SCALE DO
            ROW_TOTAL(.I.):=ROW_TOTAL(.I.) + CHI(.I,J.);

        FOR J:= 0 TO OUTPUT_SCALE DO
            FOR I:= 0 TO INPUT_SCALE DO
            COL_TOTAL(.J.):=COL_TOTAL(.J.) + CHI(.I,J.);

        FOR I:= 0 TO INPUT_SCALE DO
            FOR J:= 0 TO OUTPUT_SCALE DO
            BEGIN
              EXPECT(.I,J.):=ROW_TOTAL(.I.)*COL_TOTAL(.J.)/
                             NO_OF_TEST;
              IF EXPECT(.I,J.) = 0
              THEN BEGIN
                     WRITELN; WRITE(' EXPECT = 0 AT ',I,J);
                   END;
            END; (* BEGIN *)

        FOR I:= 0 TO INPUT_SCALE DO
            FOR J:= 0 TO OUTPUT_SCALE DO
            C_SQUARE := C_SQUARE+(CHI(.I,J.)-EXPECT(.I,J.))**2/
                        EXPECT(.I,J.);

        DEGREE_OF_FREEDOM := INPUT_SCALE*OUTPUT_SCALE;
        END; (* CHISQ *)
```

```
{*****************************************************************
 *}
        PROCEDURE OUT_DATA;
{*
 *****************************************************************
 *}
{*      THIS PROCEDURE SUPPORTS THE MAIN PROGRAM TO OUTPUT THE
 *    RELEVENT STATISTICS. THE CONTINGENCY TABLE IS ONLY PRINTED
 *  ON REQUEST.
 *
 *****************************************************************
 *}
        VAR
          I,J,INDEX : INTEGER;
        BEGIN

{*      *** THE SUITABLE HYPOTHESIS IS OUTPUTED ...              ***
*}
        IF CONST_KEY = 1
        THEN BEGIN
                WRITELN;
                WRITE ('0 HYPOTHESIS : FOR A FIXED KEY, A CHOSEN ');
                WRITE ('BUT FIXED PORTION OF CIPHERTEXT IS ');
                WRITELN;WRITE('                          ');
                WRITE ('INDEPENDENT TO A CHOSEN BUT FIXED ');
                WRITE ('PORTION OF THE PLAINTEXT.');

                WRITELN;
                WRITE ('0 FIXED KEY : ');
                INDEX := 1;
                FOR I:= 1 TO 8 DO
                     BEGIN
                       FOR J:= 1 TO 8 DO
                            BEGIN
                              WRITE (KEY[.INDEX.]:1);
                              INDEX := INDEX + 1;
                            END; (* FOR *)
                       WRITE (' ':1);
                     END; (* FOR *)
             END
        ELSE BEGIN
                WRITELN;
                WRITE ('0 HYPOTHESIS : FOR A FIXED PLAINTEXT, ');
                WRITE ('A CHOSEN BUT FIXED PORTION OF THE CIPHERTEXT ');
                WRITELN;WRITE('                          ');
                WRITE ('IS INDEPENDENT TO A CHOSEN BUT FIXED ');
                WRITE ('PORTION OF THE KEY.');

                WRITELN;
                WRITE ('0 FIXED PLAINTEXT : ');
                INDEX := 1;
                FOR I:= 1 TO 8 DO
                     BEGIN
                       FOR J:= 1 TO 8 DO
                            BEGIN
                              WRITE (PLAINTEXT[.INDEX.]:1);
                              INDEX := INDEX + 1;
                            END; (* FOR *)
                       WRITE (' ':1);
                     END; (* FOR *)
             END;
```

```
{*      *** THE CONTINGENCY TABLE IS ONLY PRINTED ON REQUEST,    ***
        *** FOR EXAMPLE, BY SETTING THE VARIABLE PRINT_TABLE = 1***
*}
          IF PRINT_TABLE = 1
          THEN BEGIN
                 WRITELN; WRITE('0 CONTINGENCY TABLE :');

                 WRITELN; WRITE('-          I');

                 FOR I:= 0 TO OUTPUT_SCALE DO
                     WRITE(I:3);
                 WRITE('    TOTAL');
                 WRITELN; WRITE ('     ');

                 FOR I:= 1 TO 5 DO
                     WRITE ('_');
                 WRITE ('I');

                 FOR I:= 9 TO 4*OUTPUT_SCALE+7 DO
                     WRITE ('_');
                 WRITELN; WRITE ('           I');
                 WRITELN; WRITE (' ');

                 FOR I:= 0 TO INPUT_SCALE DO
                     BEGIN
                       WRITE (I:6,'  I');
                       FOR J:= 0 TO OUTPUT_SCALE DO
                           WRITE (CHI[.I,J.]:3);
                       WRITE (ROW_TOTAL[.I.]:6);
                       WRITELN; WRITE ('           I');
                       WRITELN; WRITE (' ');
                     END;
                 WRITE('     ');
                 FOR I:= 1 TO 5 DO
                     WRITE ('_');
                 WRITE ('I');

                 FOR I:= 9 TO 4*OUTPUT_SCALE+7 DO
                     WRITE ('_');
                 WRITELN; WRITE ('           I');
                 WRITELN; WRITE ('   TOTAL I');

                 FOR I:= 0 TO OUTPUT_SCALE DO
                     WRITE (COL_TOTAL[.I.]:3);
                 WRITE (NO_OF_TEST:6);
               END;
```

```
{*      *** STATISTIC INFORMATIONS ARE PRINTED ****          ___          ***
*}
        WRITELN;
        WRITE ('0 SUM OF THE FREQUENCIES FOR ALL CELLS  : ',
               NO_OF_TEST:5);

        WRITELN;
        IF CONST_KEY = 1
        THEN BEGIN
             WRITE ('0 POSITIONS OF THE CHOSEN BUT FIXED PORTION ');
             WRITE ('OF THE PLAINTEXT:    ');
             END
        ELSE BEGIN
             WRITE ('0 POSITIONS OF THE CHOSEN BUT FIXED PORTION ');
             WRITE ('OF THE KEY:          ');
             END;

        FOR I:= 1 TO INPUT_TEST_BIT DO
            WRITE (ROW_TEST_BIT[.I.]:3);

        WRITELN;
        WRITE ('0 POSITIONS OF THE CHOSEN BUT FIXED PORTION ');
        WRITE ('OF THE CIPHERTEXT:   ');

        FOR I:= 1 TO OUTPUT_TEST_BIT DO
            WRITE (COL_TEST_BIT[.I.]:3);

        WRITELN;
        WRITE ('0 DEGREE OF FREEDOM :          ',DEGREE_OF_FREEDOM:5);

        WRITELN;
        WRITE ('0 SIGNIFICANCE LEVEL:          ',SIGNIFICANCE_LEVEL
              :10:4);

        WRITELN;
        WRITE ('0 REJECTION REGIONS :          ',TABLE_VALUE:10:4);

        WRITELN;
        WRITE ('0 CHI SQUARE :                 ',C_SQUARE:10:4);

        WRITELN;
        WRITE ('0 CONCLUSION FROM THE TEST ABOUT THE HYPOTHESIS: ');

        IF (C_SQUARE - TABLE_VALUE >= 0)
        THEN BEGIN
               WRITE (' THE HYPOTHESIS OF INDEPENDENCE IS');
               WRITE (' REJECTED. ');
             END
        ELSE BEGIN
               WRITE (' THE HYPOTHESIS OF INDEPENDENCE IS');
               WRITE (' ACCEPTED.');
             END;

    END; {* OUT_DATA *}
```

```
BEGIN  (* MAIN PROGRAM *)

  CPU_1 := CLOCK;
  INITIALIZATION;
  READ_DATA;
  FOR I:= 1 TO NO_OF_TEST DO
      BEGIN
        DES(I);

        IF CONST_KEY =1
        THEN LOCATION(PLAINTEXT, ROW, 0)
        ELSE LOCATION(KEY, ROW, 0);


        LOCATION(CIPHERTEXT, COL,1);
        CHI(.ROW, COL.) := CHI(.ROW, COL.) +1;
      END; (* FOR *)
  CHISQ;
  OUT_DATA;

  CPU_2 := CLOCK;
  WRITELN;
  WRITE ('0 TOTAL CPU TIME USED : ',(CPU_2 - CPU_1)/1000
            :10:4,' SECONDS. ');

END. (* MAIN PROGRAM *)
```

REFERENCE

ADLE79 Adleman, L., "A subexponential algorithm for the discrete logarithm problem with applications to cryptography", Proc. of the Twentieth IEEE Symposium on Foundations of Computer Science, (Oct. 1979), 55-60.

BALL80 Ball, A.J.S., Bochmann, G.V., and Gecsei, J., "Videotex networks", IEEE Trans. on Consumer Electronics, Vol.CE-25, No.3, (Dec. 1980), 8-14.

BARK77 Barker, W.G., Cryptanalysis of the Hagelin Cryptograph, Laguna Hill, CA: Aegean Park Press, (1977).

BEYE Beyer, W.H., Handbook of Tables for Probability and Statistics, second edition, The Chemical Rubber Co.

BOOT81 Booth, K.S., "Authentication of signatures using public key encryption", Comm. ACM, Vol.24, No.11, (Nov. 1981), 772-774.

BRAN76 Branstad, D.K., Gait, J., and Katzke, S., "Report of the workshop on cryptography in support of computer security, "National Bureau of Standards, NBS IR 77-1291, (Sept. 1976), 21-22.

BRIG77 Bright, H.S., "Cryptanalytic attack and defense: ciphertext-only, known-plaintext, chosen-plaintext", Cryptologia, Vol.1, No.4, (Oct. 1977), 366-370.

BRYC81 Bryce, H.D., "Data security", IEEE Trans. on Consumer Electronics, Vol.CE-27, (May 1981), 159-165.

DATA82 Data Star, "Unit encrypts eight bytes in 25 microseconds", IEEE Computer, (June 1982), 103.

DAVI79 Davida, G.I., "Hellman's scheme breaks DES in its basic form", IEEE Spectrum, Vol.6, No.7, (July 1979), 39.

DAVI80a Davies, D.W., and Price, W.L., "The application of digital signatures based on public-key cryptosystems", Proc. of the Fifth ICCC, (Oct. 1980), 525-630.

DAVI80b Davies, D.W., and Price, W.L., "Selected papers in cryptography and data security", NPL Report, (Nov. 1980), 1-22.

DAVI82 Davies, D.W., Tutorial: The Security of Data in Networks, IEEE Computer Society, (1982).

DENN81 Denning, D.E. and Sacco, M., "Timestamps in key distribution protocols", Comm. ACM, Vol.24, No,8, (Aug. 1981), 533-536.

DIFF76 Diffie, W. and Hellman, M.E., "New directions in cryptography", IEEE Trans. on Information Theory, Vol.IT-22, No.6, (Nov. 1976), 644-654.

DIFF77 Diffie, W., and Hellman, M.E., "Echaustive cryptanalysis of the NBS data encryption standard", Computer, Vol.10, No.6, (June 1977), 74-84.

DIFF79 Diffie, W. and Hellman, M.E., "Privacy and authentication: an introduction to cryptography", Proc. IEEE, Vol.67, No.3, (March 1979), 397-427.

EDWA66 Edwards, D.L., "OCAS on line cryptanalytic aid system", M.I.T. Project MAC Technical Report #27, (May 1966).

EHRS78 Ehrsam, W.F., Matyas, S.M., Meyer, C.H., and Tuchman, W.L., "A cryptography key management scheme for implementing the data encryption standard", IBM Syst. J., Vol.17, No.2, (1978), 106-125.

FEIS58 Feistel, H., "A survey of problems and systems in authenticated communication and control", M.I.T. Lincoln Lab., (May 1958).

FRIE44 Friedman, W.F., Military Cryptanalysis, Washington, DC: U.S. Government Printing Office, (1949), 4 volumes, I-Monoalphabetic Substitution Systems. II-Simpler Varieties of Polyalphabetic Substitution Systems, III-A periodic Substitutions, IV-Transposition Systems. (available in several major libraries).

GAIT56 Gaines, H.F., Cryptanalysis, A Study of Ciphers and Their Solution, New York: Dover, (1956).

GAIT77a Gait, J., "A new nonlinear pseudorandom number generator", IEEE Trans. on Software Eng., Vol.SE-3, No.5, (Sept. 1977), 359-363.

GAIT77b Gait, J., "Validating the correctness of hardware implementations of the NBS data encryption standard", National Bureau of Standards, Special Publication.

GARD77 Gardner, M., "A new kind of cipher that would take millions of years to break", Sci. American, Vol.237, (Aug. 1977), 120-124.

GROS74 Grossman, E., "Group theoretic remarks on cryptographic systems based on two types of addition", IBM T.J. Watson Research Centre, Yorktown Heights, NY, RC4742, (Feb. 1974).

GUDE80 Gudes, E., "The design of a cryptography based secure file system", IEEE Trans. on Software Eng., Vol.SE-6, No.5, (Sept. 1980), 411-420.

HEAT76 Heaton, D.L., and Wright, H.O., "LSI implementation of the proposed data encryption standard", Presented at the Nat. Computer Comf., New York, NY, (June 1976), 7-10.

HELL76 Hellman, M.E., Merkle, R., Schroeppl, R., Washington, L., Diffie, W., Pohlig, S., and Schweitzer, P., "Results of an initial attempt to cryptanalyze the NBS data encryption standard", Stanford Univeristy, Stanford, CA, SEL76-042, (Sept. 1976).

HELL79 Hellman, M.E., "DES will be totally insecure within ten years", IEEE spectrum, Vol.16, No.7, (July 1979), 32-39.

HERL78 Herlestam, T., "Critical remarks on some public-key cryptosystems", BIT, Vol.19, (1979), 274-275.

KAHN76 Kahn, D., The Codebreakers, the Story of Secret Writing, MacMillan, New York, (1976).

KENT81 Kent, S.T., "Security requirements and protocols for a broadcast scenario", IEEE Trans. on Comm., Vol.COM-29, No.6, (June 1981), 778-786.

KEY 76 Key, E.L., "An analysis of the structure and complexity of non-linear binary sequence generators", IEEE Trans. Information Theory, Vol.IT-22, (Nov. 1976), 732-736. KOLA77 Kolata, G.B., "Computer encryption and the national security agency connection", Science, Vol.197, (July 1977), 438-440.

KULL76 Kullback, S., Statistical Methods in Cryptanalysis, Aegean Park Press, Laguna Hills, Calif., (1976).

LEMP79 Lempel, A., "Cryptology in Transition", Computer Surveys, Vol.11, No.4, (Dec. 1979), 286-303.

HOFF77 Hoffman, L.J., Modern Methods for Computer Security and Privacy, Prentic-Hall, Inc., (1977).

MART79 Marti, B., Poignet, A., Schwartz, C., and Michon, V., "The Antiope videotex system", IEEE trans. on Consumer Electronics, Vol.CE-25, No.3, (July 1979), 327-332.

MATY78 Matyas, S.M., and Meyer, C.H., "Generation, distribution, and installation of cryptographic keys", IBM Syst. J., Vol.17, No.2, (1978), 126-137.

MERK78a Merkle, R.C., "Secure communications over insecure channels", Comm. ACM, Vol.21, No.4, (April 1978), 294-299.

MERK78b Merkle, R.C., and Hellman, M.E., "Hiding information and signatures in trapdoor knapsacks", IEEE trans. on Information Theory, Vol.IT-24, No.5, (Sept. 1978), 525-530.

MERK80 Merkle, R.C., "Protocols for public key cryptosystems", Proc. of the 1980 symposium on security and privacy, IEEE Computer Society, (1980), 122-123.

MORR77 Morris, R., Sloane, N.J.A., and Wyner, A.D., "Assessment of the National Bureau of Standards proposed federal data encryption standard", Cryptologia, Vol.1, (July 1977), 281-291.

NEED78 Needham, R.M. and Schroeder, M.D., "Using encryption for authentication in large networks of computers", Comm. ACM, Vol.21, No.12, (1978), 993-999.

NBS 77 National Bureau of Standards, "Data encryption Standard", FIPS 46, (Jan. 1977), 1-18. Also in Tutorial on Computer Society, (1977), VII.14-28.

OBRI82 O'brien., G.D., Bown, H.G., smirle, J.C., Lum, Y.F., and Kukulka, J.Z., Telidon Videotex Presentation level Protocol: Augmented Picture Description Instructions, Communications Research Centre, Ottawa, Canada, (1982).

POHL78 Pohlig, S.C., and Hellman, M.E., "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance", IEEE Trans. on Information Theory, Vol.IT-24, No.1, (Jan. 1978), 106-110.

POPE79 Popek, G.J. and Kline, C.S., "Encryption and secure computer networks", Computer Surveys, Vol.11, No.4, (Dec. 1979), 331-356.

PRIC81 Price, W.L. and Davies, D.W., "Issues in the design of a key distribution centre", NPL Report, (April 1981).

RIVE78a Rivest,R.L., Shamir, A., and Adleman L., "A method for obtaining digital signatures and public-key cryptosystems", Comm. ACM, Vol.21, No.4, (Feb. 1978), 120-126.

RIVE78b Rivest, R.L., "Remarks on a proposed cryptanalytic attack on the M.I.T. public-key cryptosysem", Cryptologia, (Jan. 1978), 62-65.

RIVE79 Rivest, R.L., "Critical remarks on some public-key cryptosystems by T. Herlestam", BIT, (1979), 274-275.

ROSC69 Roscoe, J.T., Fundamental Research Statistics; Holt, Rinehart and Winston, Inc., (1969).

SEND78 Sendrow, M., "Key management in E.F.T. environments", Proc. COMPCON, (Sept. 1978).

SHAM80 Shamir, A., and Zippel. P.E., "On security of the Markle-Hellman cryptography scheme", IEEE Trans. on Information Theory, Vol.IT-24, No.1, (Jan. 1978), 339-340.

SIMM77 Simmons, G.J., and Norris, M.J., "Preliminary comments on the M.I.T. public key cryptosystem", Cryptologia, Vol.1, (Oct. 1977), 406-414.

SIMM79 Simmons, G.J., "Symmetric and asymmetric encryption", Vol.11, No.4, (Dec. 1979), 305-330.

SMID81 Smid, M.E., "Integrating the data encryption standard into computer networks", IEEE Trans. on Comm., Vol.COM-29, No.6, (June 1981), 762-772.

SINK68 Sinkov, A., Elementary Cryptanalysis, a Mathematical Approach, New Mathematical Livrary #22, Random House, New York, (1968).

STEP78 Stephan, E., "Communication standards for using DES", Proc. COMPCON, (Sept. 1978).

SYKE77 Sykes, D.J., "Implementation of the NBS encryption standard", presented at the 1977 Honeywell Computer Security and Privacy Symp., (April 1977), 19-20.

TANE81 Tanenbaum, A.S., Computer Network, Pentice-Hall, Inc., (1981), 387-436.

TROU80 Troughton, D.P., "Prestel operational strategy", Proc. Videotex, Viewdata, Teletext, 1980, 51-62.

TUCH78 Tuchman, W.L., "Efficacy of DES in data processing", Proc. COMPCON, (Sept. 1978).

TUCH79 Tuchman, W., "Hellman presents, no shortcut solutions to the DES", IEEE Spectrum, Vol.16, No.7, (July 1976), 40-41.

TUCK73 Tuckerman, B., "Solution of a substitution-fractionationtransposition cipher", IBM T.J. Watson Research Centre, Yorktown Heights, NY. RC4537, (Sept. 1973).

WIDM76 Widmer, W.R., "Message authentication, a special identification requirement in one-way digital data transmission", Proc. Int. Zurich Seminar on Digital Communications, (March 1976).

WOOL80 Woolfe, R., Videotex, Heydem & son Ltd., (1980).

YASA76 Yasaki, E.K., "Encryption algorithm: key size is the thing", Datamation, Vol.22, No.3, (March 1976), 164-166.