United States General Accounting Office

GAO

Program Evaluation and
Methodology Division

March 1987

Privacy Data: The Data Encryption Standard Provides Valuable Protection

Transfer Paper 8

PREFACE

This paper addresses the rationale for and uses in evaluation of data encryption procedures with particular emphasis on the data encryption standard (DES). The data encryption standard provides a tool that can greatly increase the privacy and confidentiality of sensitive data. By being able to provide this protection an evaluator may more easily obtain personal or otherwise sensitive data. Specifically, program evaluators now have a means for protecting any sensitive information they have collected.

This paper is not applicable to situations in which the potential user has an application that involves national security information. Further, it is not applicable to some civil uses--such as those involving electronic funds transfers--that will require a certified implementation of the DES, which requires a hardware rather than a software process. In other cases, the DES software process presented here may provide a relatively simple and economical method of protecting the privacy of data. After reading this paper, evaluators should be able to

--recognize when data encryption may be useful in a study,

--understand the basic approach to utilizing data encryption in dealing with privacy or confidentiality concerns, and

--provide the technical documentation so that a GAO design, methodology, and technical assistance group in headquarters, a technical assistance group from the regions, or their counterparts outside GAO could construct and operate a computer program to perform data encryption.

Examples of general applications are discussed in order to provide an evaluator with a good basis on which to judge the utility of encryption in specific settings. However, certain guiding assumptions can be shared immediately:

--The data encryption standard provides a benchmark for encryption work because of its status as an official federal standard. The reader is cautioned, however, that DES may not be used to safeguard national security information.

--Protection of identity, as well as the preservation of a unique identifier for data retrieval or matching, can be provided by enciphering the personal identifier, either name or number.

--Encrypted data files should have some form of unencrypted backup. That is, the evaluator should be able to recreate the encrypted data if necessary.

With these assumptions guiding the applications of data encryption efforts, the evaluator is in an advantageous methodological position to undertake work that previously may have been only partially completed because of concerns with privacy and confidentiality.

Privacy Data: The Data Encryption Standard Provides Valuable Protection is one of a series of papers issued by PEMD. The purpose of the series is to provide GAO evaluators with handy, clear, and comprehensive guides to various aspects of evaluation methodology and explain specific applications. Other papers in the series include Causal Analysis: A Method to Identify and Test Cause and Effect Relationships in Program Evaluations, Content Analysis: A Methodology for Structuring and Analyzing Written Material, Designing Evaluations, Using Structured Interviewing Techniques, Statistical Sampling, and Developing and Using Questionnaires.

Readers of this paper are encouraged to send questions or comments on it to me.

Eleanor Chelimsky
Director

Contents

ABBREVIATIONS

ACM	Association for Computing Machinery
ASCII	American standard code for information interchange
DES	Data encryption standard
EBCDIC	Extended binary code for data interchange
ECB	Electronic codebook
FIPS	Federal information processing standard
IBM	International Business Machines
MAC	Message authentication code
NBS	National Bureau of Standards
NSA	National Security Agency

CHAPTER 1

INTRODUCTION

This chapter discusses an evaluator's need to protect data and introduces a software cryptographic approach to data protection, the data encryption standard (DES), an official federal standard recommended for broad classes of protection.[1] The DES can be implemented on a broad range of computers, both large and small. Moreover, after the technical details of implementation have been carried out, the encryption may be used with no further requirement for technical skills. It can be used to protect identities, personal information, or any other data that needs to be protected. (Examples of its use are discussed in chapter 2.)

The DES is a cryptographic method that is in the public domain. The sensitive data are protected by being cryptographically enciphered. The outcome of the encipherment depends on the data being protected and by a user-selected key, one of approximately 72 quadrillion choices. The safety of the sensitive data is ensured by the protection of the cryptographic key, generally an easier task than the protection of the original data.

NEED TO PROTECT DATA

There are situations when certain types of information need to be physically safeguarded from becoming available to those outside some designated group. As almost everyone knows, national security information must be protected against accidental disclosure. There is also some proprietary or otherwise privileged information that likewise must be restricted. GAO evaluators must be able to provide adequate security and protection to such data if we are to fulfill our pledges of confidentiality.

---

[1]Cryptography renders a message unintelligible to outsiders by various transformations of symbols used to transmit the message. The plaintext is the uncoded message that is to be put into secret form. Ciphertext is the transformed (encoded) plaintext. (The words encipher, encode, and encrypt are used interchangeably.) A key, or keyword, is a (secret) set of symbols (numbers or characters) selected by the user to initialize the encryption and thus generate the protected ciphertext. A block, as used in this paper, is a group of 8 consecutive characters, e.g., "12345678" or "Now is t." The standard for the software cryptographic approach discussed in this paper, DES, is the following publication: U.S. Department of Commerce, National Bureau of Standards, Data Encryption Standard (Washington, D.C.: January 1977), federal information processing standard (FIPS) publication 46.

In addition, we need to consider Privacy Act safeguards. The Privacy Act of 1974 was an attempt by Congress to legislate general governmentwide standards for the protection of individual privacy by imposing restrictions on the collection, maintenance, use, and dissemination of personally identifiable records. This act, as well as any private agreement an evaluator may have entered into (e.g., pledges of confidentiality), spells out the necessity for protecting personal information. Although as a legislative agency, GAO is exempt from the Privacy Act, we attempt, as a matter of policy, to carry out our responsibilities in a manner consistent with the spirit of the act. In addition, because we frequently use other agencies' privacy systems of records in our work, good practice requires that personal information be carefully safeguarded both against public release and inappropriate uses. This paper addresses data encryption as a tool for protecting privacy and confidentiality.

METHODS OF PROTECTION

Generally speaking, the greatest protection is afforded by ensuring physical security and controlling access to the information to be protected. Very often in the storage and transmittal of secret data, security will be afforded through physical means such as safes, armed couriers, and shielded transmission cables. Indeed, physical protection is still valuable in this age of data processing and electronic communications. When messages are hand-carried, there are no signals to intercept.

However, couriers and hand-delivered documents can generate their own communication problems, e.g., timeliness. A document that is hand-carried from one place to another takes longer to get to its destination than one which is transmitted by electronic means. Electronic transmission of information, however, pays a price for speed--the accessibility of the message. Anyone with the appropriate receiving equipment can intercept the message.

Data encryption enables users to protect their data by rendering it incomprehensible to any person who is not able to undo the encryption. Generally speaking, cryptography finds methods to transform messages (the plaintext) into cryptograms (the ciphertext). Some encryption methods are intended only to afford limited temporal protection. Others, while theoretically vulnerable, in practice offer invulnerable protection.

DATA ENCRYPTION STANDARD

The data encryption standard specifies an algorithm to be implemented in electronic hardware devices and used for the cryptographic protection of computer data. It was adopted by the National Bureau of Standards (NBS) on July 15, 1977. As such, the DES is a federally approved standard for safeguarding the transmission and storage of all data that are not classified according to the National Security Act of 1947, as amended, or the

Atomic Energy Act of 1954, as amended. Use of the DES is encouraged by nonfederal government organizations when such use provides the desired security for commercial and private organizations.[2]

Very briefly, the DES works by using an enciphering key selected by the user to initialize a pattern of transpositions and substitutions.[3] The data to be enciphered are subjected to the transformations and the output of the process is the ciphertext. The DES has a very attractive and useful feature in that (for a given key) unique inputs will result in unique outputs. Since the operation is symmetric, this means that two identical outputs must have come from identical inputs; see, for example, table 2.1 on p. 16.

The security provided by the DES algorithm is based on the fact that an unauthorized recipient of a message (intercepted) must have the entire key in order to decipher the encrypted data. That is, if the interceptor were to attempt deciphering the message using an "almost correct" key, the correct message would not be recovered from the encrypted data. This is true even if only one of the 56 binary digits in the key is incorrectly specified. It is an all or nothing system.

---

[2]For example, the Federal Reserve System has an active encryption policy for all messages transmitted within the system. In addition, it is pursuing the use of the authentication mode of the DES. That is, all messages sent over the system will have to have an authenticator block, even if they are already encrypted. (The purpose of an authenticator is to detect message tampering. Use of an authenticator does not require that the message itself be transmitted in enciphered form, nor does it reveal if the message has been intercepted. It does protect against an undetected alteration in the message.) The Department of the Treasury requires that all existing systems that accomplish electronic fund transfers with the Treasury Department must incorporate an authenticator by 1988. All new electronic fund transfer systems must incorporate such an authenticator immediately. For further information on this policy, see the following document: U.S. Department of the Treasury, Directives Manual (Washington, D.C.: August 16, 1984), policy statement number 8180, ch. TD81, sec. 80.

[3]The key, which is selected by the person performing the encryption, determines the precise sequence of transformations used in the encryption process. Although any string of symbols, i.e., alphanumeric characters, could be used as a key, a properly constructed DES key must meet a parity constraint. Refer to the section entitled "Specification of a DES Keyword" on page 26 for an easily followed "cookbook" approach to the construction of a randomly selected key.

The only known way of obtaining the key with certainty is by obtaining matched ciphertext and plaintext and then by exhaustively testing keys by enciphering the known plaintext with each key and comparing tne result with the known ciphertext. In principle, a special purpose computer could be constructed to perform these tests sequentially and by so doing, recover the key. However, since 56 independent bits are used in a DES key, 2^{56} such tests are required to guarantee finding a particular key. The expected number of tests to recover the correct key is 2^{55}. Thus, at one microsecond per test 1142 years would be required to recover the correct key.[4]

DES hardware and software

Technically, the DES is a hardware standard. That is, the encryption algorithm is in accord with the standard only when it is implemented in a physical electronic circuit. Software applications are not certified applications of the DES. The standard specifically states: "Software implementations in general purpose computers are not in compliance with this standard."[5] However, this is important only to those individuals or organizations whose policies require them to be in compliance with the official NBS standard. Software implementations of the DES provide the same degree of protection to the data after they are encrypted. However, it is very easy to have an incorrect implementation. In fact, if the software has met the validation tests, the resulting ciphertext will be identical to that produced by a hardware implementation, so long as the same key and text blocking is used.[6]

ADVANTAGES AND DISADVANTAGES OF THE DES

The DES is increasingly being established as the de facto American nonmilitary encryption standard, it will probably be adopted by the International Standards Organization.

The DES has been subjected to numerous attempts at deciphering over the last ten years and there have been no known methods of attack developed other than the "brute force" technique of trying all possible keys.

---

[4]U.S. Department of Commerce, National Bureau of Standards, Guidelines for Implementing and Using the NBS Data Encryption Standard (Washington, D.C.: April 1981), FIPS 74, p. 9.

[5]Data Encryption Standard, p. 2.

[6]J. Gait, Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard, U.S. Department of Commerce, National Bureau of Standards, NBS special publication 500-20 (Washington, D.C.: September 1980).

This data protection, however, is dependent on the physical protection of the encryption key. If the key is available, all documents enciphered with that key are potentially compromised.

The uniqueness of the encipherment is also an advantage. The user of the data does not have to worry about two distinct input records having identical enciphered identifiers. If there is a need to go back to the original data base entries, the deciphering leads back to the original unique identifier. That is, the uniqueness of the identity keys is preserved by the encipherment. At the least, this can ease an analyst's concern over allowing for duplicate identifiers.

The DES affords the ability to match data from different files while preserving privacy and anonymity so long as the owners of the files (data bases) are willing to share encryption keys or the use of a common look-up table.[7]

An additional advantage lies in a lessening of the record keeping burden placed on the owner of the population data. The only information the data base owner has to retain (and protect) is the key used for the encipherment. If the owner wishes to maintain a list of the records provided, he or she may do so. However, such a list is not necessary because the enciphered identifier is carried with each record. Any follow-up only requires the decipherment of the identifier, clearly an improvement over having to use record books to look up the identity of each respondent based on some arbitrary identifier in the record.

This improved efficiency of record keeping also applies to those cases when it would be useful to draw additional cases from the file to extend the original sample. The data base owner can just extract a new random sample of whatever type is required and provide the new (enciphered) sample to the user. The user then has only to remove duplicates and run the analyses.

Another advantage of the DES over manual methods is its speed. Some observers feel that the eventual preferred communications system may incorporate two encryption systems: the DES for the secure transmission of data proper and another type of system, which will be used only for the key-management portion of the transmission.

---

[7]Even in the case when the data base owners are not willing to share keys, there are strategies that permit the matching of protected data. See, for example, R. F. Boruch, "Strategies for Eliciting and Merging Confidential Social Research Data," Policy Sciences, 3 (1972), 275-97. There is always a need for some amount of trust, even when a "broker" is proposed, but the DES as an official standard can strengthen the case for protected sharing of information.

However, in spite of the speed of the DES, encryption does impose an overhead burden on a project, albeit not too serious. See appendix III for representative equipment costs. The data to be encrypted must be laid out and resources obtained to perform the encryption. A somewhat more serious potential disadvantage of the DES follows from its strength. Encrypted data should have some type of unencrypted backup. That is, the evaluator should not maintain an encrypted file as the only data source. If the key is lost, or the encryption is faulty, the data are truly lost.

Any software encryption scheme is vulnerable to "spoofing," that is, an unauthorized, subtle change to an otherwise correct implementation. The result of such a change is that the user thinks encryption is being performed but, in reality, the protection is only apparent and not real. One way to guard against this possiblity is to incorporate a protection scheme into the code implementation itself. This would reveal, or prevent, the attempt at tampering.

Additionally, a software tool should be totally self-contained so as to maximize its utility. That is, the potential user should be able to just specify data to be protected and "flip a switch" to perform the encryption.

The implementations presented in this paper are not totally self-contained packages in that the source code must be recompiled each time before the algorithm is used. This is also a strength, however, because it is more difficult to make unauthorized and undetected changes to source code than to a compiled machine language program. This is especially true when the implementation may be validated each time before it is used in a specific application (see page 30).

Finally, there are two areas that need to be mentioned in connection with the DES. Neither area affects the use of the DES, per se, but both areas raise related questions. First, it may not be possible to match files if the owners are unwilling to share some amount of information. Second, there are many open questions about the extent to which personal information needs to be either purged or camouflaged to prevent identification through data relationships such as date of birth, city, occupation, and employer.

WHO SHOULD USE THIS REPORT

This report is intended for use by GAO evaluators and auditors, with assistance from a Design, Methodology, and Technical Assistance Group (DMTAG) or other technical experts in GAO who are knowledgeable about FORTRAN or "C" programming. The evaluator should be able to encrypt data from executive agencies with assistance from these individuals, along with a description of the data to be encrypted and a copy of this report.

A FORTRAN or "C" compiler is necessary for installation of this encryption tool on a computer mainframe. It can also be installed on a computer mainframe at an executive agency, where the data are encrypted and then stored on a computer tape. The basic electronic codebook mode may be applicable when the data to be encrypted are 8 characters long, or fewer; the cipher block chaining mode may be appropriate for encryption of entire files or any data that are more than 8 characters in length.

The data encryption standard used in this software tool is not appropriate for systems of "classified national security information" as well as "other sensitive, but unclassified information, the loss of which could adversely affect the national security interest," as stated in national security decision directive (NSDD) 145. Military data, classified data, and other information related to the national security interest must be encrypted and protected using means other than those provided in this transfer paper. The GAO uses of the data encryption standard could include, for example, privacy data and "sensitive, but unclassified" data such as Social Security numbers, Internal Revenue Service files, health care services evaluation information, and pension and retirement data.

THE ORGANIZATION OF THIS PAPER

Chapter 1 of this paper discusses the need to protect data and introduces the data encryption standard (DES), one of the official standards adopted under the provisions of public law 89-306 (Brooks Act) and under part 6 of title 15, Code of Federal Regulations.

Chapter 2 discusses uses of the DES in protecting data in various settings and in merging data obtained from different sources. Such uses are illustrated by the use of the DES in a GAO study that dealt with personnel records of senior officers in the U.S. Marine Corps.

Chapter 3 presents for a technical audience, such as a headquarters DMTAG or a regional TAG, a technical overview and discussion of a software encryption program that simulates the hardware operation of the DES. The chapter also presents a simple method for the random selection of a keyword to be used by the encryption program.

Appendix I contains detailed listings and flowcharts of the FORTRAN program used to validate the performance of the system that is performing the DES encryption.

Appendix II presents examples of several short main programs. These would be substituted in place of the validation tests to produce customized encryption programs that might be used in GAO evaluations.

Appendix III contains a detailed listing of a DES program written in C language. This program would be used to perform file encryption for storage or transmission.

Appendix IV briefly mentions other means of performing data encryption.

Appendix V discusses recent developments in data encryption and the anticipated emergence of new hardware encryption device designed by the National Security Agency for national security information.

Following appendix V are the bibliography and the glossary.

CHAPTER 2

USES OF THE DATA ENCRYPTION STANDARD

This chapter begins by discussing DES modes of operation and then goes on to consider the following uses of the DES as a method of cryptographic data protection:

--Protection of personal identifiers to ensure privacy.

--Protection of sensitive data during electronic transmission.

--Protection of sensitive data in computer files.

--Ensuring the confidentiality of data.

--Ensuring against unauthorized changes to data (message authentication).

--Merging personal data from different sources.

Ensuring the confidentiality of data and merging personal data from different sources have, we believe, the greatest potential benefit to GAO. Our capacity to ensure confidentiality should facilitate obtaining sensitive and potentially restricted data in GAO evaluations. Being able to merge data from different sources means we can enhance the potential of those data we do obtain.

Future opportunities will exist to make use of the DES in GAO. For example, agencies such as the Internal Revenue Service, the Social Security Administration, the Office of Personnel Management, and the Department of Defense personnel units might be more willing to release data if the DES is used to encrypt all appropriate personal information.

If two of these groups elected to share a DES key, the GAO evaluator would be able to obtain a "blind" matching sample from, say, the Social Security Administration and the Office of Personnel Management.

MODES OF OPERATION

The DES has four distinct modes of operation which specify how data will be encrypted and decrypted. The modes are the electronic codebook mode, the cipher block chaining mode, the cipher feedback mode, and the output feedback mode.[1] Basically,

---

[1]U.S. Department of Commerce, National Bureau of Standards, DES Modes of Operation (Washington, D.C.: December 1980), FIPS 81.

they involve differences in how the encryption algorithm will be propagated between subsequent pieces of data. The greater part of this paper addresses the electronic codebook mode, a mode that was tested and used by GAO in a comparatively easy to apply FORTRAN software implementation. The cipher feedback and output feedback modes are not discussed in this paper.

Electronic codebook mode

In the electronic codebook mode, each block of information is enciphered independently of the result of the previous block's encipherment. That is, given the same key, the word "evaluate" will be identically encrypted wherever it occurs. This mode is well suited to the encryption of names or numbers that serve as individual identifiers. A Social Security number that is used to identify an individual occurs only in the context of an identifier. It is part of a larger record, but it has a separate existence. It is worth pointing out, however, that in terms of cryptographic security this is the weakest of the DES operating modes.

Cipher block chaining

The paper also provides a listing of a DES program written in the C language. The greater speed of C, as compared to FORTRAN, makes this version of the DES algorithm better suited for implementing the cipher block chaining (CBC) mode. As the name suggests, this mode feeds, or "chains," the output of each block encipherment into the algorithm for the next block. Thus the code text for a given clear text block depends on both the key and the preceding block.

In particular if one set of two blocks was "PROGRAM," "ANALYSIS," and a second set was "SPECIAL," "ANALYSIS," the word "ANALYSIS" would have two different cipher text representations--even using the same key. This is because the first set used "PROGRAM," which influenced the encryption, while the second set used the block "SPECIAL."

This mode is particularly well suited to the encryption of entire records or files. The unit of encryption is in, a sense, the entire file. Two files will be identically encrypted by a single key only if they are identical files. It is time, however, that two records which start with the same data, will be enciphered identically so long as the clear text is the same. Two messages which are the same at the end are enciphered into different ciphertext. See appendix III for a listing of the C program.

If an evaluator were enciphering an entire list of identifiers this might be the method of choice. For individual Social Security members it should provide the same ciphertext as the electronic codebook mode. That is, if there is no second block, there is no chaining to take place.

The workings of the DES algorithm in the electronic codebook and cipher block chaining modes are illustrated in figures 2.1 and 2.2. In the left-hand column, a user has a message to protect (the plaintext). The plaintext is structured into an input block and fed into the DES algorithm together with a user-selected keyword. The encryption process transforms the input block into an output block, which, when written out, becomes the ciphertext. The decryption process is an identical process. The ciphertext is formatted into an input block and passed to the DES decryption algorithm. There it is decrypted and processed into an output block which becomes the (recovered) plaintext. In addition, figure 2.2 illustrates the chaining process.

Figure 2.1: DES Algorithm
(Electronic Codebook Mode)

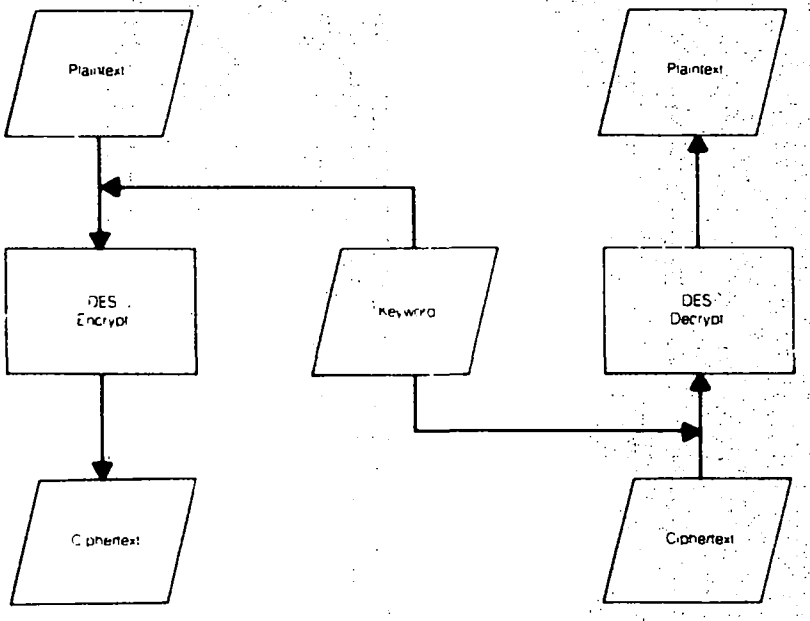
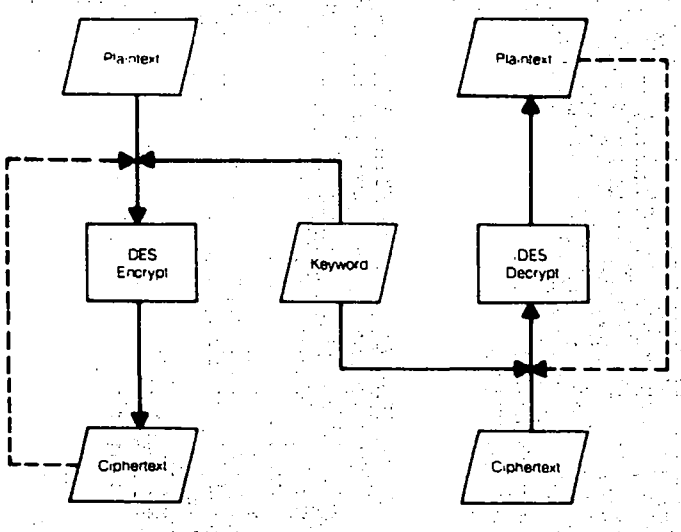


Figure 2.2: DES Algorithm
(Cipher Block Chaining Mode)



Much of the power of the DES algorithm is its ability to camouflage (i.e., protect) data coupled with its simplicity of use. This power is illustrated in table 2.1. Notice how two nearly identical Social Security numbers are changed into very different ciphertext by the same key (the columns of the table).[2] Also note how two nearly identical keys will encipher the same identifier into very different ciphertext (rows 1 and 2).

Table 2.1

Changes in Ciphertext Resulting From Plaintext and Key Changes

Social Security number	Key 1 (34627691FEDCAB08)	Key 2 (54627691FEDCAB08)
587 52 7365	0534092AEA6E716F	AE426AD25F7987F4
587 52 7366	5E4FDC5EEBFFF8D6	FD2DF378DB55208C

PROTECTION OF SENSITIVE DATA

All agencies have data that they wish to protect against unauthorized access and disclosure. GAO is no different. In some cases the information to be protected consists of entire documents, while in others it suffices to protect only certain pieces of information within each document.[3] The amount of protection will necessarily vary with the perceived sensitivity of the information. In order to decide on the amount of protection to provide, these questions must be addressed:

1. Is it necessary to retain the original document?

2. How much information must be protected in each document?

---

[2]The keys and ciphertext are written in hexadecimal notation. The mathematical manipulations of the DES lead to the use of hexadecimal numbers. See page 28 for a table of binary-hexadecimal-decimal equivalents.

[3]Because of restrictions on use of the DES, this discussion does not apply to materials classified under either the National Security Act of 1947, as amended, or the Atomic Energy Act of 1954, as amended. Obviously, the DES also does not apply if the information to be protected is the original paper document. Thus it is assumed in what follows that it is not important to save original documents. That is, only the information is important, not what it is recorded on.

3. Why is the document being enciphered?

Five examples of particular interest to GAO are summarized next.

Protection of personal files for privacy purposes

The cryptographic community refers to the "privacy problem" in a technical sense as preventing an opponent from extracting information from a communication channel. If the concept of a communication channel is extended to include personal information in a folder filed in a cabinet or personal information in a computerized system of records, it will also apply to what most people consider to be the primary privacy concern--that information collected on them is only used for the purposes that they are officially aware of and is not diverted to other uses. (Note that this includes unauthorized access as a special case.) Again, it is assumed that information is to be protected, not physical documents as such.

In the simplest application, a given record (document) has only the name (or possibly the Social Security number) of the individual to be protected. That is, it is assumed there are no additional identifiers (or combinations of identifiers) that could reveal the identity of the individual to someone who had access to that record in the file. In this case, the DES can be used to protect identity by enciphering the personal identifier--either name or numeric. After the encipherment, the record has the same information content as before, but the individual's identity is only available to those who can decipher the "scrambled" identifier. Again, unique identifiers in the original record (plaintext) become unique identifiers in the enciphered record (ciphertext).

An application of this type, with only a limited amount of information to protect, is well suited to the use of a computer software routine such as that illustrated in appendix I to provide the encipherment. As the amount of data to be protected increases, the hardware implementations of the DES become more attractive. The speed of encipherment is very high so that is feasible to encipher all the data in the document.[4]

The main attractiveness of the DES for privacy protection lies in its very simplicity. Records could always be protected by stripping out personal identifiers and replacing them with, for example, an index number. However, the list of index numbers must be physically protected and might be a quite large document. If

---

[4]Integrated circuits that could encipher over 250,000 characters per second were in use by IBM in 1978.

two or more groups are to share access to the protected records, the indices must also be shared. This imposes certain administrative burdens on all concerned. Using encipherment for protection still requires physical protection of the key, but it is much simpler to protect a single 8- or 16-character key than an entire index document. In order for two parties to share access to the data, only the key need be shared.[5]

Transmission of data between audit staffs

The protection of data during transmission is one of the tasks for which the DES was originally intended. Depending on the volume of information to be protected, a software or hardware implementation of the DES could be used. If the total amount of information to be transmitted were small, then a software implementation could be reasonable. It is the total amount of information to be protected that is relevant, not how much is on each document. There are a number of manufacturers who can supply "black boxes" to encipher all data that flow through them. The user need only select an operating mode and the key to be used. There are also software packages available that will encipher a computer data file. The enciphered file could then be transmitted electronically in protected form to its destination.

As stated earlier in this paper, national security (classified) information should not be transmitted using the DES, but almost anything else is a possible candidate. A prime candidate is "sensitive" information that is needed quickly. For example, a revision of a chapter in a draft report could be enciphered and transmitted so long as there were some procedure for coordinating the key. (It should be possible to set up a key schedule at the start of a job, and just change the keys used, say, on the first of each month.) Proprietary data obtained from contractors is also a good candidate for protection. If the data

---

[5]A "weakness" of all single-key encipherment systems is that the key must be shared (and protected) by all parties who need access to the deciphered data. This is not really a problem when a large number of parties require equal access. It can be a problem if there are a large number of parties who only share in pairs, because each pair needs a separate key. An approach called public-key cryptosystems addresses the concern of key transmission and the more general problem of message authentication. See, for example, D. Chaum, "Security Without Identification: Transaction Systems to Make Big Brother Obsolete," Communications of the ACM, vol. 28:10 (October 1985), 1030-44.

were needed for, say, responding to agency comments, there would be a premium on its rapid receipt. Personal information or budgetary data could likewise be protected in transmission by means of the DES. As in the earlier discussion of personal information the amount of information to be protected is dependent on its sensitivity. At one extreme, only personal identifiers need be enciphered; at the other, the entire document.

Transmission of data between user and computer complex

In a like manner, data to be transmitted between a computer user and a central computer complex could be enciphered for protection. At the least, this would protect the data in transit. If the raw data were not deciphered at the central site, any attempt at analysis would be futile. Protection at the main computer, per se, is more difficult because of the need to have the key available to the central computer. Even though different data items would still be distinct, there would be no metric that could transfer from the ciphertext to the plaintext. The only measure that could be calculated would be the mode (if it existed).

The actual transmission would most likely be accomplished by means of a specialized communication device that incorporated the DES as an electronic microcircuit, for example, a special circuit board added to a microcomputer. Data would go through the encryption circuit before being transmitted by the microcomputers. Such a device could work in concert with a similar unit at the central site, thereby preserving data integrity during transmission.

Storage of sensitive information in computer files

The DES can be used to encipher data files that are to be stored in a computer. The storage medium can be permanent as in disk packs on a central computer or hard disks on a microcomputer, or it can be removable as in floppy disks and tape drives. In permanent storage media protection will depend on keeping the key separate from the data and making use of the key only when actually making use of the data. That is, the data are generally maintained in their enciphered form. They are deciphered only when they are actually being used, and are then re-enciphered.

An alternative to the complete encipher and decipher cycle each time the data are accessed would be to attach a "reverse state" index to each record. That is, select the key that will be used to encipher the data and use it to "decipher" a set of indices that are, in fact, already plaintext. These ciphertext indices are then attached to the appropriate records. Because of the symmetry of the DES, when the file is enciphered these

ciphertext indices will revert to readable plaintext.[6] This, in a sense, is close to the best of all possible worlds. The data file is protected by the DES while, at the same time, it contains a plaintext index that can be used for record location and retrieval. In this way, only the record(s) being retrieved need to be deciphered and the associated overhead is substantially reduced.

Message authenticator

The DES has an additional application, that of data, or message, authenticator. According to FIPS 81,

> "A Message Authentication Code (MAC) is generated (computed) as a cryptographic function of the message (data). The MAC is then stored or transmitted with the data. Only those knowing the secret key can recompute the MAC for the received message and verify that the message has not been modified by comparing the computed MAC with the stored or transmitted MAC. An unauthorized recipient of the data who does not possess the key cannot modify the data and generate a new MAC to correspond with the modified data. This technique is useful in applications which require maintaining data integrity but which do not require protecting the data from disclosure. For example, computer programs may be stored in plain text form with a computed MAC appended to the program file. The program may be read and executed without decryption. However, when the integrity of the program is questioned, a MAC can be computed on the program file and compared with the one stored in the file. If the two MAC's are identical and the cryptographic key used to generate the MAC has been protected, then the program file has not been modified."[7]

---

[6]This exploits the property of the DES that the encipher and decipher operations are formal inverses of each other. Given the same key, deciphering is accomplished by running the ciphertext through the algorithm starting at what was the last transformation and ending with what was the first transformation when the plaintext was enciphered.

[7]DES Modes of Operation, p. 24. The message authentication code is also referred to as a data authentication code. This use is further defined in U.S. Department of Commerce, National Bureau of Standards, Computer Data Authentication (Washington, D.C.: May 1985), FIPS 113.

A very important use of the MAC is verifying the correctness of enciphered binary data, or more generally, any numeric data. Suppose, for example, binary data (a computer data file, say) are encrypted and somehow a block is garbled. When the garbled block is decrypted, the outcome will be garbled data but it will not readily be recognized as being in error--after all, it will still be a binary pattern. If, however, a MAC is used, you will at least know that a garble has occurred and are warned that there is a problem with the data.

MERGING DATA FROM DIFFERENT SOURCES

It is frequently the case that two different data sources have information on a common population. The degree of overlap varies widely and is a property of the data sources. That is, the researcher may be able to select the data sources, but he or she will have to accept the structure and information content of each data base. In some cases, one data set could be a subset of the other. For example, one file might contain death certificates from all causes of death for a given region and a second file might contain records of claims filed under workman's compensation. While one would not expect all claims to be matched with death certificates, there is some expectation that a portion of the names in the claim file will match with the death data. In other cases, the population might be nearly the same in the two files, but the information collected could be quite different. Two illustrations of this application follow.

Women, Infants, and Children (WIC) study

In a 1979 GAO study of the WIC program, the maternal health records were maintained by one program office, the family data by another.[8] The two offices did not have a common file. Because the data were considered privileged medical information--i.e., not releasable--GAO was unable to obtain records in a form that would permit them to be linked and analyzed.

If the DES had been available at that time, and if the two offices were willing to share use of an encryption key or a look-up table, identifiers could have been encrypted and used to link records from the different offices, thus building a much richer data base for analysis. That is, given the two sets of data linked through the enciphered identity of the mother, it might

---

[8]GAO (U.S. General Accounting Office), The Special Supplemental Food Program for Women, Infants, and Children (WIC): How Can It Work Better?, CED-79-55 (Washington, D.C.: February 1979).

have been possible to draw statistical conclusions about infant and maternal health using variables from the different files. In particular, the evaluators might have been able to reach conclusions that could not be reached from information contained within one single file and that could not be established without the several variables being positively linked through a specific individual. Although at the time of this study a tool such as the DES was not available, there was a clear requirement for a method that could be used to perform such linkages.

U.S. Marine Corps study

The 1984 GAO study of the quality of retained senior Marine Corps Officers involved administrative data in the Marine Corps headquarters master file and performance records from their automated fitness reporting system.[9]

The Marine Corps administrative archive contained a variety of administrative information on all Marines. Data items included an identifier (Social Security number), personal information such as race and marital status, and professional information such as academic degrees and military occupational specialties. It did not contain the records of the performance evaluations that Marines received throughout their careers.

The performance records, or fitness reports, were maintained separately and had performance evaluation information for each Marine. Data items included Social Security number, job title, name of the unit through which reports are filed, and assessments of the performance of items such as regular duties, handling officers, tactical handling of troops, personal appearance, presence of mind, and growth potential, as well as general value to the service.

These two data bases were maintained by separate offices. Information from them was joined together only for the confidential use of promotion boards. GAO's study needed information from both data sources in order to derive and validate a metric of "quality." The Marine Corps was concerned about both privacy and confidentiality. The Corps felt that even though GAO was exempt from the requirements of the Privacy Act of 1974, notification that data were provided to GAO would still have to be placed in the personnel files of all officers about whom data were given to GAO. In addition, the performance data are very closely held. An individual Marine may request a copy of his or her own file and turn the information over to whomever he or she wishes, but a second party may not receive copies of the performance rating directly.

---

[9]GAO (U.S. General Accounting Office), High-Quality Senior Marine Corps Officers: How Many Stay Beyond 20 Years of Service?, GAO/PEMD-85-1 (Washington, D.C.: November 1984).

This was another example of a requirement to be able to match data from different sources. The difference between the previously cited WIC study and this one was that by this time the DES had become available. Personnel at the National Bureau of Standards provided GAO with a computer program implementing the DES algorithm. After modifications to obtain the speicific tool needed to meet the requirement at hand, GAO extensively tested and reverified the validation of the software to perform the DES encryption.

GAO documented the official status of the DES and provided a FORTRAN computer program incorporating the DES algorithm to the Marine Corps. The Marine Corps personnel staff felt that use of the DES in the manner outlined by GAO would be sufficient to address their concerns about privacy and confidentiality.

GAO guided the Marine Corps staff responsible for the administrative file in the use of GAO's DES computer package to encode each officer's Social Security number. This encoded number replaced the actual number in the administrative records furnished to GAO. That office then provided the paired Social Security and enciphered numbers to the staff maintaining the performance data. They similarly stripped out personal identifiers and substituted the enciphered identifiers. Thus they were able to provide GAO with actual but anonymous records.

While preserving privacy and confidentiality, the DES encoding was useful also in "data cleaning." That is, when there was a problem with apparently inconsistent data, GAO was able to ask the Marine Corps to decode the numbers used as identifiers, track down the original information, and correct the data.

CHAPTER 3

TECHNICAL DESCRIPTION: HOW TO USE

DES SOFTWARE ENCRYPTION

This chapter is a guide to the technical reader who would be responsible for implementing the algorithm. It presents the steps necessary to use the DES software algorithm. The chapter includes

--a brief overview of the data encryption processing flow,

--a method for specifying a DES keyboard, and

--suggestions for using the DES emulation computer program, that is, tailoring the software to specific needs.

OVERVIEW OF THE DATA ENCRYPTION PROCESSING FLOW

The DES works by successively manipulating information (plaintext) that is composed of 64 consecutive binary digits. For most purposes this 64-bit string can be visualized, or thought of, as a string of 8 bytes (characters).

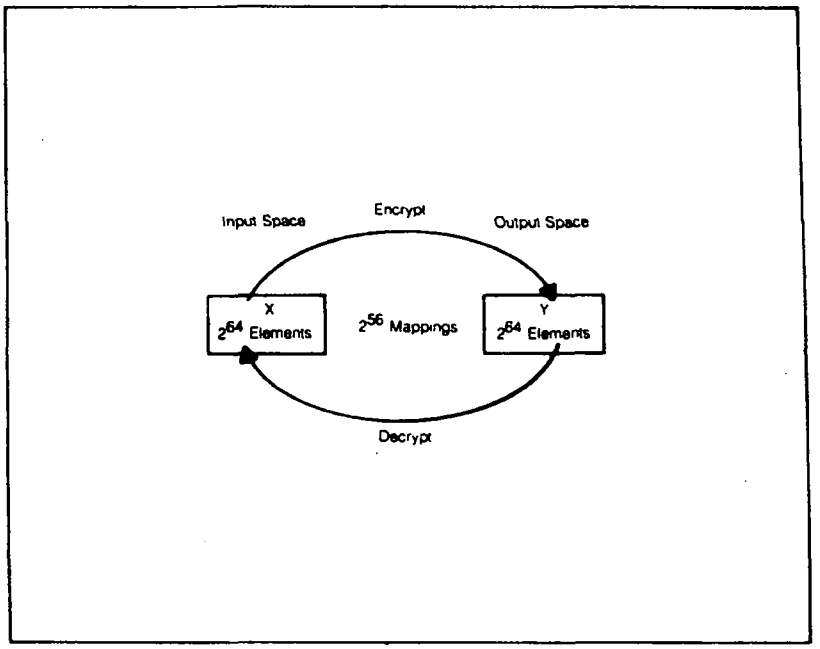
Mathematically, the DES maps the set of all possible 64-bit vectors onto itself (see figure 3.1). The DES cryptographic key allows the user to select any one of 2^{64} possible invertible mappings--i.e., transformations that are one-to-one. Selecting a key selects one of these mappings. When using the DES in the electronic codebook mode with a particular key, each input is mapped onto a unique output in encryption and this output is an iterative, block, product cipher system. This means that it alternates between transposition and substitution operations and uses the output of one operation as the input to the next. The DES internally uses 16 iterations of a pair of transposition and substitution operations to encrypt or decrypt an input block.[1]

The electronic codebook mode is a basic, block, cryptographic method that transforms 64 bits of input to 64 bits of output as specified in FIPS 46. The analogy to a codebook arises because the same plaintext block always produces the same ciphertext block for a given cryptographic key. Thus a list (or codebook) of plaintext blocks and corresponding ciphertext blocks theoretically could be constructed for any given key. In an electronic implementation, the codebook entries are calculated each time for the plaintext to be encrypted and, inversely, for the ciphertext to be decrypted.

---

[1]The material presented here is taken from DES Modes of Operation, pp. 12-13. It has been edited to varying degrees.

Figure 3.1: Number of Possible DES Codes



Each bit of an output block in this mode is a complex function of all 64 bits of the input block and all 56 independent (nonparity) bits of the cryptographic key.[2] Thus a single bit error in either a ciphertext block or the nonparity key bits used for decryption will cause the decrypted plaintext block to have an average error rate of 50 percent. However, an error in one ciphertext block will not affect the decryption of other blocks. That is, there is no error propagation between blocks in this mode.

Since this mode is a 64-bit block cipher, an electronic codebook device must encrypt data in integral multiples of 64 bits. If a user has less than 64 bits to encrypt, then the least significant bits of the unused portion of the input data block must be padded, that is, filled with random or pseudorandom bits, prior to encryption. The corresponding decrypting device must then discard these padding bits after decryption of the ciphertext block. The same input block always produces the same output block under a fixed key in this mode.

---

[2]Because the DES operates on binary input to produce binary output, visual examination of the output will require looking at binary strings or their equivalent. One commonly used equivalent is the hexadecimal (base 16) representation. If the computer, or display device, can support hexadecimal representation, then all possible binary strings may be shown as 16-digit hexadecimal numbers rather than 64-bit binary strings.

Specific details about the flow of data through the transpositions and substitutions boxes (referred to as "S-boxes" in the literature), together with the manner in which the key is combined with the plaintext may be found in FIPS 46. It is essentially correct to say that the message (input) is permuted, broken into two parts, and then subjected to a complex set of transformations that depend on (i.e., vary with) both the key and the message.

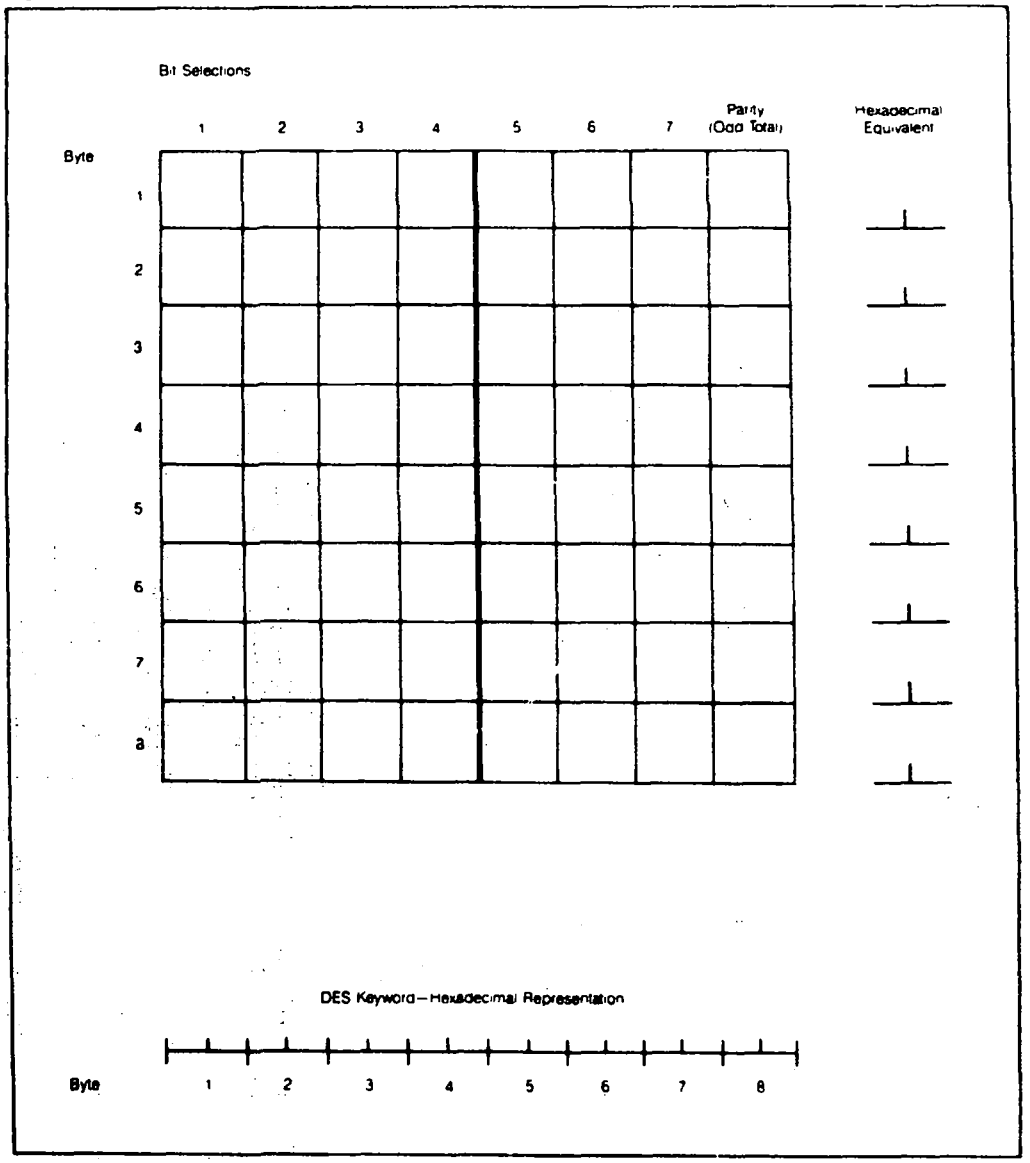
SPECIFICATION OF A DES KEYWORD

Use of the DES requires the selection of a keyword for the encryption and subsequent decryption processes. The keyword itself is a 56-bit binary string that is expanded to a 64-bit string by incorporating a parity bit after each 7 bits. Each resulting 8-bit byte should have odd parity.

Since a randomly generated keyword offers the greatest security, the following procedure is offered as a fast, unbiased method of generating random numbers:

1. Obtain 7 coins, paper, and a pencil.
2. Shake coins in cupped hands.
3. Without observing states ("heads" or "tails"), place coins on table and keep covered with hand.
4. Extract one coin from the group and record its state--e.g., "heads" is "1" and "tails" is "0." (Optional: record state of coin on DES keyword recording form, figure 3.2) Repeat for each of the next 6 coins.
5. Select the parity bit so that the total number of "1's" in the 8-bit byte is odd (i.e., including the parity bit itself).
6. Repeat steps 1-4 to generate seven more bytes.
7. Translate successive 4-bit groups into their hexadecimal representation and use the resulting 16-digit hexadecimal number as the DES keyword.

Figure 3.2: DES Keyword Recording Form



USE OF THE DES EMULATION COMPUTER PROGRAM

The DES emulation program as documented in this paper is configured to accept several input formats: a 9-digit number (e.g., a Social Security number), a 16-digit hexadecimal number, and an 8-character string.[3] The last 2 inputs could produce identical results when used on data stored as internal binary representations on a machine-readable computer file. They are normally used, however, for different purposes.

Hexadecimal input is normally used for input of either a DES key or an enciphered value. Character input would be used for reading plaintext that was physically represented as character strings. For example, the symbol "A9" is a 2-character string that is represented in IBM's EBCDIC coding as the binary string "1100000111111001."[4] The 2-character string, "A9," may value "10101001." This value is, in turn, the EBCDIC representation of the symbol "z."

The purpose of this discussion is to sensitize the reader to the fact that there are different classes of inputs. The program can manipulate the different classes, but the classes must be kept straight. A hexadecimal string that was the output of an encipherment step must be input as a hexadecimal string if it is to be deciphered. The program can accept the enciphered data as a character string and process it through the decryption steps. However, in so doing, it would not recover the original plaintext.

---

[3]The binary-hexadecimal-decimal conversion table is as follows:

Binary	Hexadecimal	Decimal	Binary	Hexadecimal	Decimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

We are indebted to Dr. Dennis Branstad, Institute for Computer Science and Technology, National Bureau of Standards, for providing us with FORTRAN source code for DES. The version presented here has been derived from Dr. Branstad's program.

[4]EBCDIC stands for "extended binary code data exchange." See glossary for further information.

The program flow is written to accept a user-defined application section in place of the tests on pages 43-46. Examples of such sections are given in appendix II.

Suppose, for example, there is a requirement to encipher a 9-digit identifier located in positions 10-18 within a 200-character record, and to do this for an arbitrary number of records. One way to do this would be to set up a small application program that read in each record as 3 fields. The first 9 characters and the last 182 characters are read in and subsequently output without any change. The identifier is read in, enciphered, and output as a 16-digit hexadecimal number. Note that, treated as a character string, the output record is now 207 characters long (9 + 182 + 16). This read-encipher-write cycle would be repeated until the input file was exhausted. Other data processing could be incorporated as a part of this cycle, or could be deferred to a separate program.

If the information to be encrypted consisted of textual information, a person's name, for example, the user application portion of the program must read in and encipher successive strings of 8 characters each until the entire character block has been processed. The logic in this case must test each string to see if it has the required 8 characters. If it does not, then it needs to be padded with randomly selected characters so that an 8-character string can be input to the DES encryption program.[5] In this case, input-output considerations are simplified if the character block to be enciphered has a fixed length. That is, any necessary padding with randomly selected characters is done outside the framework of the encryption program. (Note, however, this is not required.)

---

[5]The security of the DES in the electronic codebook mode is based on the number of inputs to the codebook. This means rather than pad with blanks, messages to be encrypted should be padded with randomly selected characters, if necessary, and the padding discarded on decryption. This type of padding maximizes the number of possible inputs to the codebook and minimizes the possibility of recurring patterns.

DES UTILITY PROGRAM

The DES Utility Main is a general-purpose encryption-decryption program that implements the DES electronic codebook mode and is designed to be used with specific user application programs. These user supplied programs will tailor data input and output and data encryption and decryption procedures to the needs of the particular study.

The particular user application detailed in this appendix is a program to simulate the validation of a DES hardware chip. Other applications are illustrated in appendix II.

Programs are written in FORTRAN IV and are configured for IBM VS FORTRAN. All programs shown have been tested on IBM 3081 and 3084 computers at the National Institutes of Health, Division of Computer Resources and Technology.

NOTES FOR FIGURE I.1, DES UTILITY MAIN

The DES Utility Main is configured to perform all aspects of the DES validation testing.[1] (See pages 43-46.) The tests themselves are performed by the Subroutine HKEY. The user program illustrated in this example is test 6, and in particular, test 6.4 within test 6. The maintenance test program creates a cycling process that tests the complete functioning of the DES algorithm as well as testing for stuck-at-one and stuck-at-zero faults at the various input or output interfaces.

"NBR" determines which of the four tests will be performed.
NBR = 64--Test 6.4: Tests all aspects of the algorithm.

Input for test 6.4

```
//STEP2  EXEC  FORVLKGO
//GO.SYSIN  DD  *
0000000000000000
6                    ---ITEST
64                   ---NBR              (Compare with
5555555555555555     ---KEYHEX            Gait, p. 55.
FFFFFFFFFFFFFFFF     ---HEXCLR
```

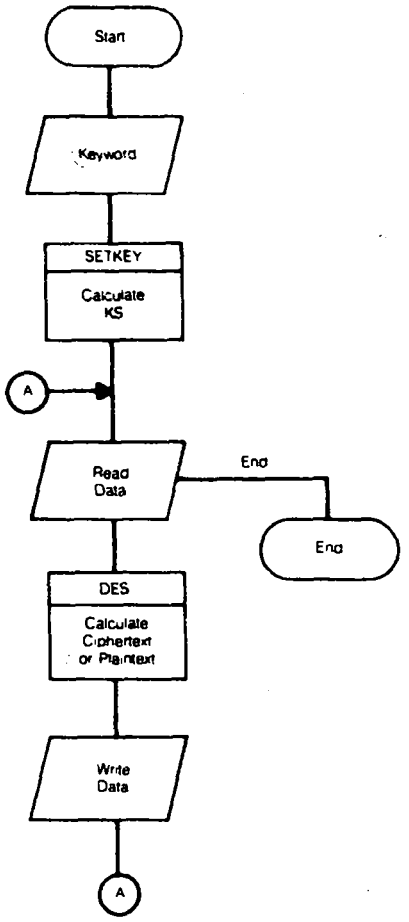
Output for test 6.4

KEYHEX	HEXCLR
246E9089C550381A	EC6AF9EE48717817

[1]Gait, p. 3.

APPENDIX I

Figure I.1: DES Utility Main



NOTES FOR FIGURE 1.2, SUBROUTINE SETKEY, AND FIGURE 1.3, SUBROUTINE SETKEY--EXPANDED

--Array KS is initialized by Subroutine SETKEY(IOPT). A key must be specified before beginning the DES. SETKEY does not have to be called again unless the user specifies a key change.

--Two options are provided for key input:

IOPT = 1	A 64-character binary representation, KEYBIN. Input format is 64I1.
IOPT = 2	A 16-character hexadecimal representation, KEYHEX. Input format is 8Z2.

--A suggested procedure for generation of a DES keyword is discussed in chapter 3, pages 26-27.

Figure I.2: Subroutine SETKEY

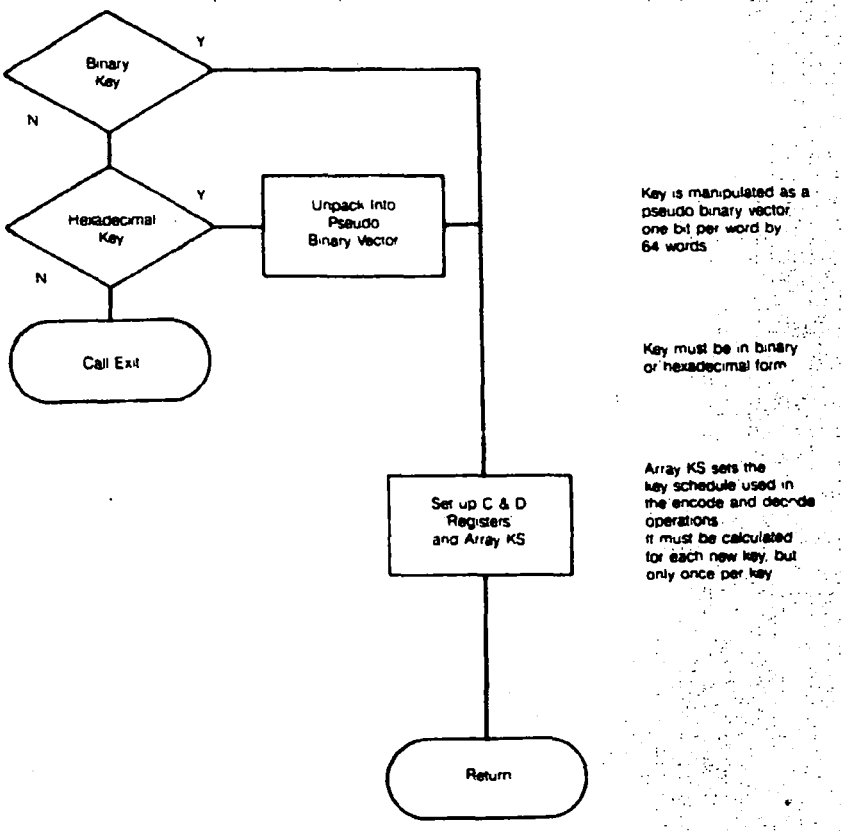
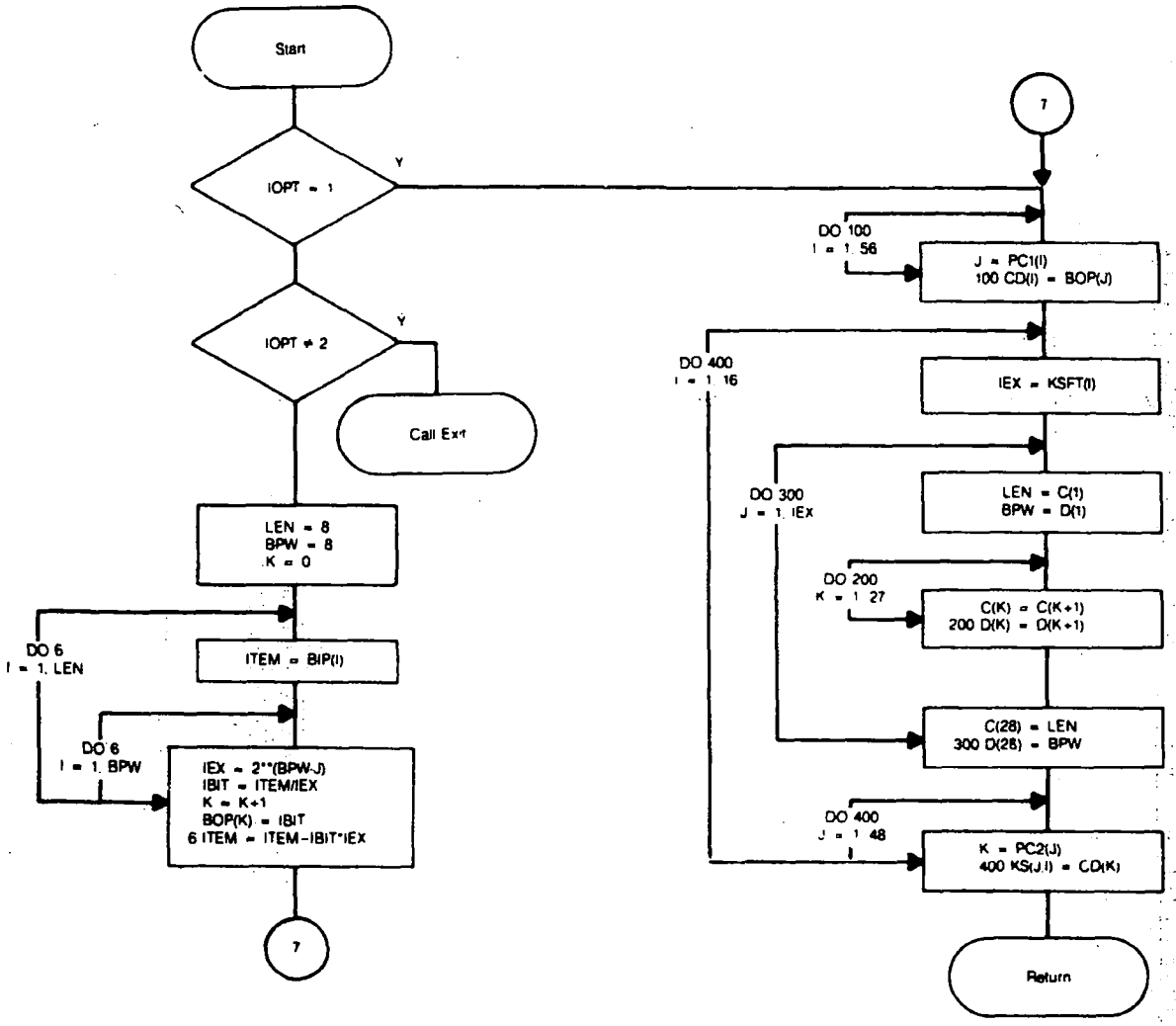


Figure I.3: Subroutine SETKEY—Expanded

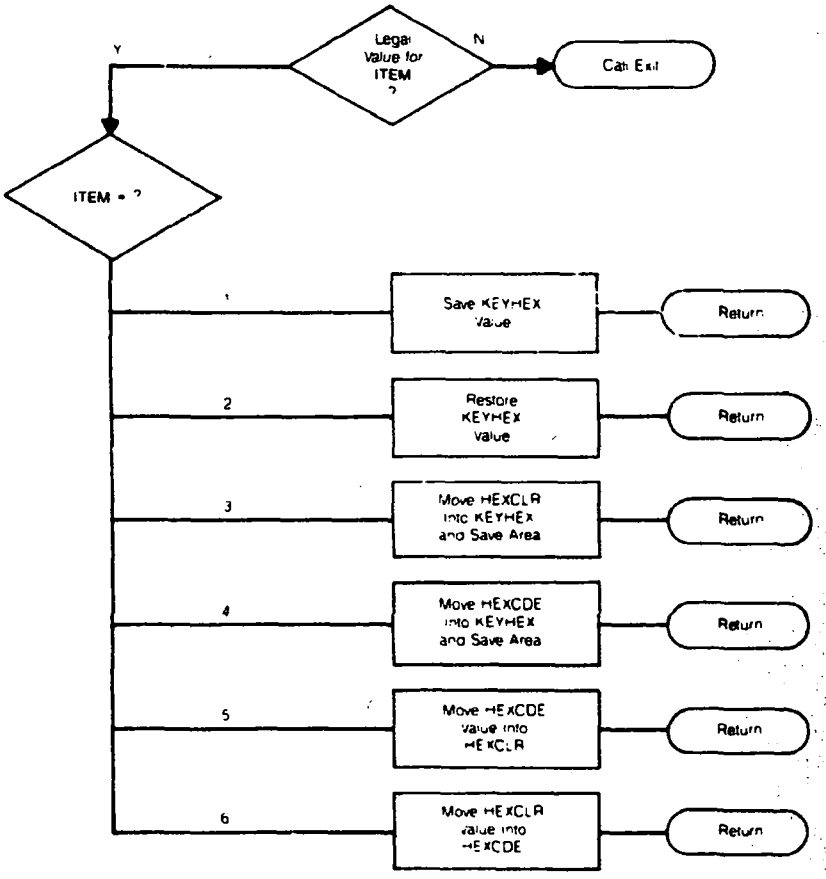


NOTES FOR FIGURE I.4, SUBROUTINE HKEY

Subroutine HKEY facilitates validation testing of the DES algorithm. Several of the tests require iterative changes of the key. Subroutine HKEY simply stores and recalls the value of an input key. (Subroutine SETKEY processing does not retain the value of the input key.)

This subroutine is a part of the user application package, in this case, the validation of the DES algorithm. It is not a part of the main DES program.

Figure I.4: Subroutine HKEY



NOTES FOR FIGURES I.5 - I.9, SUBROUTINE DES

This subroutine provides for the encryption-decryption of main program variables.

Plaintext label		Ciphertext label
IDCLR format(I7)	<===>	IDCDE format(8Z2)
SSCLR format(I9)	<===>	SSCDE format(8Z2)
HEXCLR format(8Z2)	<===>	HEXCDE format(8Z2)

Processing is sensitive to the use of these precise labels and formats. A call for encryption-decryption will generate ciphertext or plaintext for the corresponding variable without disturbing the input representation.

CALL DES(3)	Generates ciphertext "IDCDE" and "SSCDE" corresponding to the main program's "IDCLR" and "SSCLR" values.
CALL DES(4)	Generates plaintext "IDCLR" AND "SSCLR" corresponding to the main program's "IDCDE" and "SSCDE" ciphertext.
CALL DES(1)	Generates ciphertext "HEXCDE" corresponding to the plaintext "HEXCLR."
CALL DES(2)	Generates plaintext "HEXCLR" corresponding to "HEXCDE."

While DES(3) and DES(4) provide for the simultaneous encryption or decryption of two variables, they may also be used when only a single variable is present.

The processing scheme employs the electronic codebook mode of DES encryption and decryption. The translation between plaintext and ciphertext depends solely on the key input via Subroutine SETKEY, the DES algorithm, and the input representation. File sequence does not affect the operation--under a fixed key, the same input block always produces the same output block:

```
//STEP01      EXEC  FORVCOMP //COMP.SYSIN  DD    *
```

Figure I.5: Subroutine DES

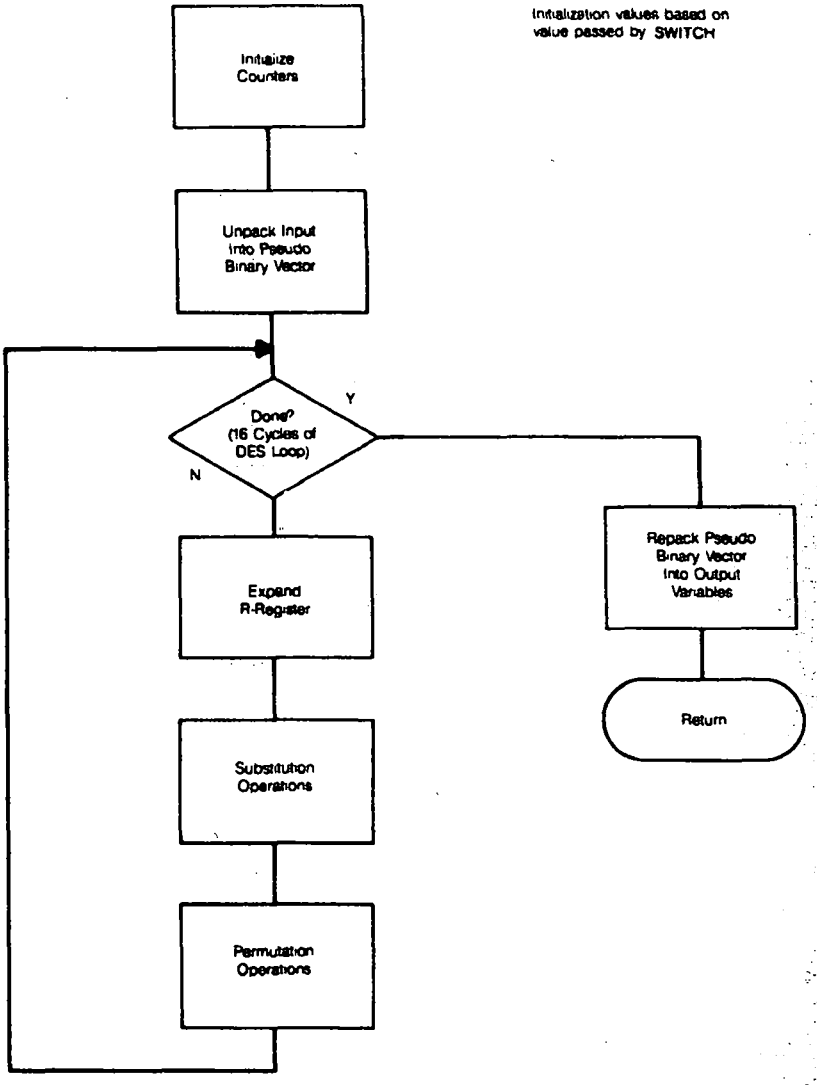
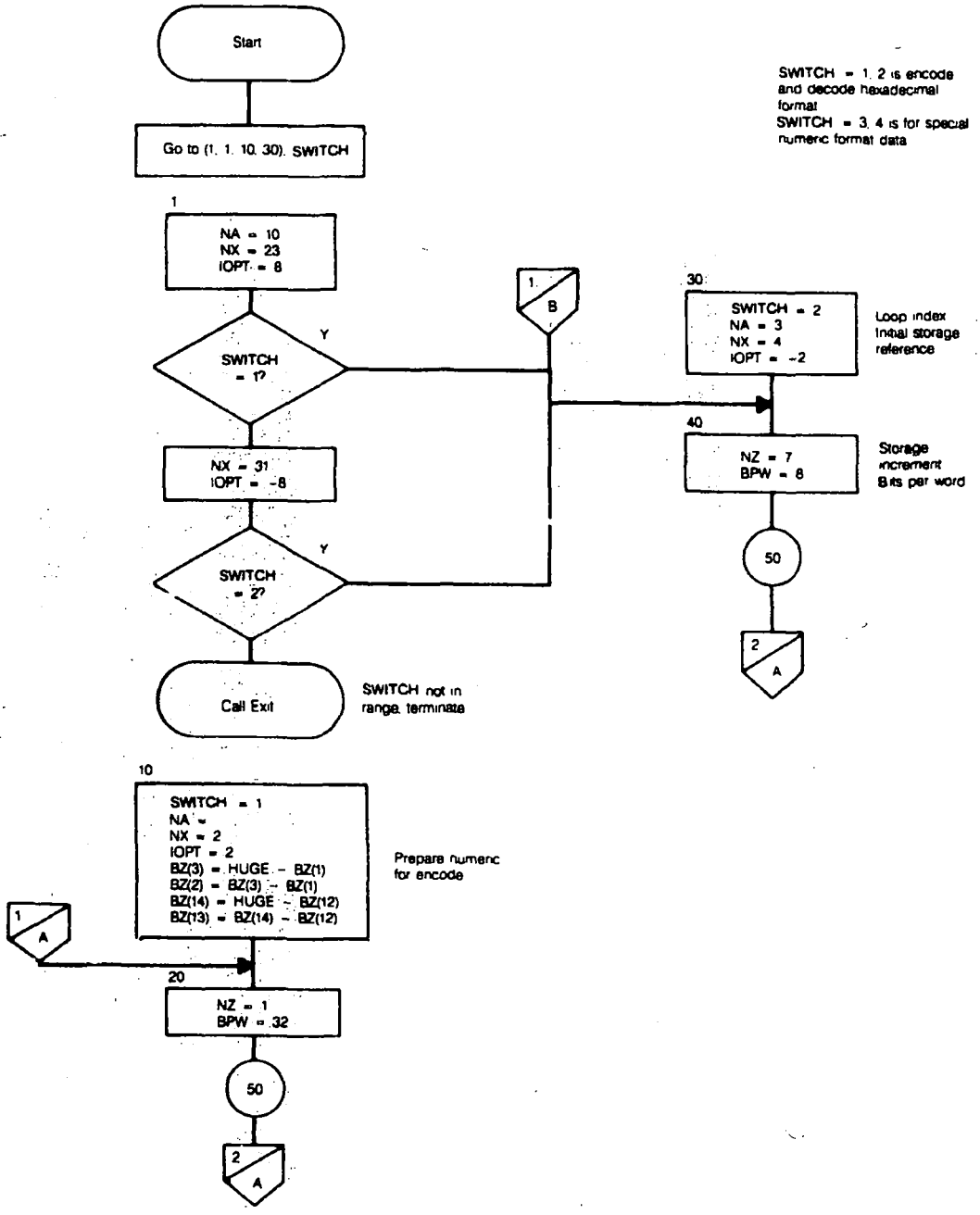
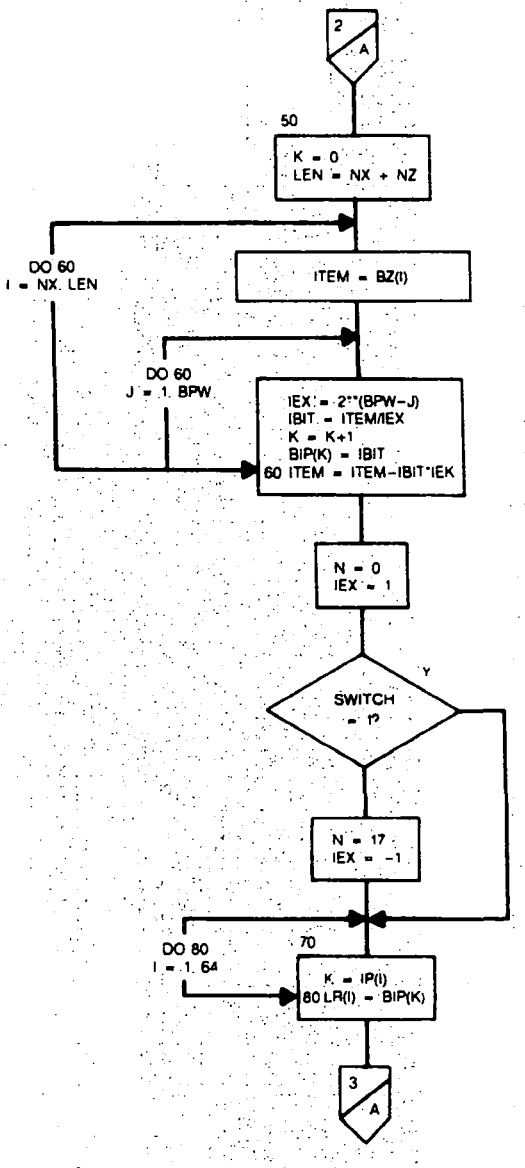


Figure I.6: Subroutine DES—Segment 1



APPENDIX I

Figure I.7: Subroutine DES—Segment 2



Unpack into 64-word pseudo binary vector

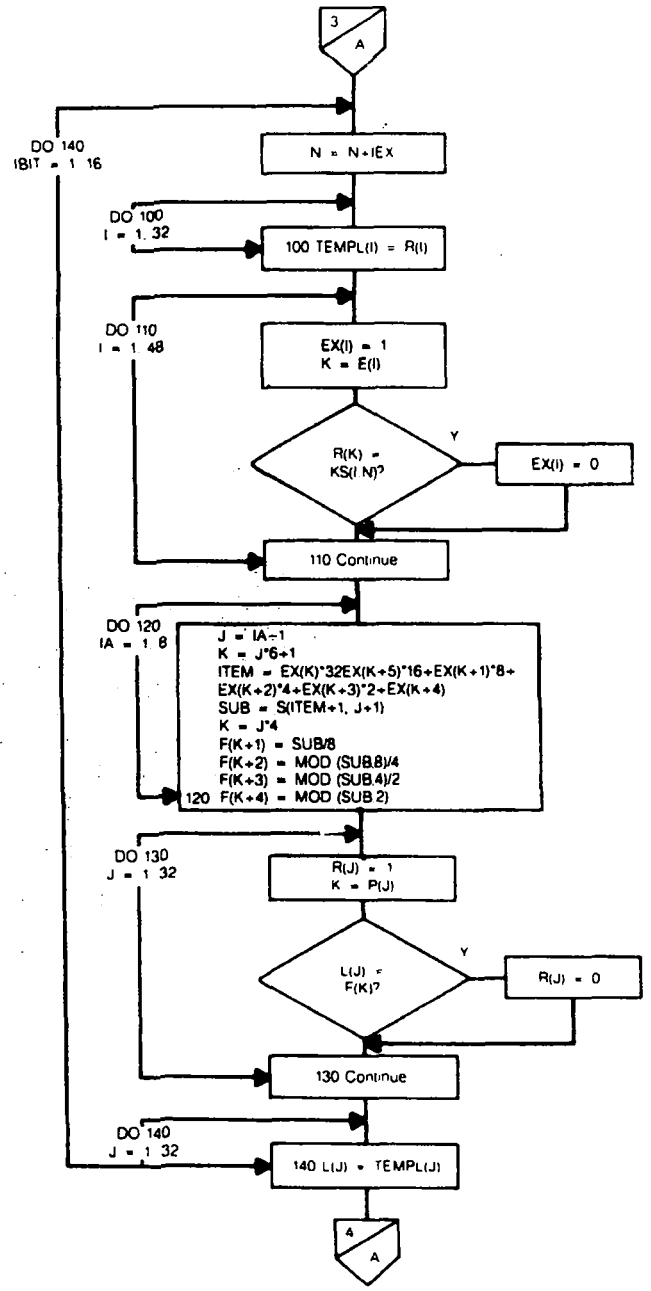
Using program common storage BZ(NX) through BZ(LEN), unpack the least significant "BPW" bits per word into a 64-word pseudo binary vector. BIP(K), K = 1, 64

Begin DES algorithm

Initial permutation of input

APPENDIX I

Figure I.8: Subroutine DES—Segment 3



Begin main loop which encrypts or decrypts for 16 rounds

Save R-register

Expand R-register using the E-function, perform exclusive OR of the key schedule and the expanded R

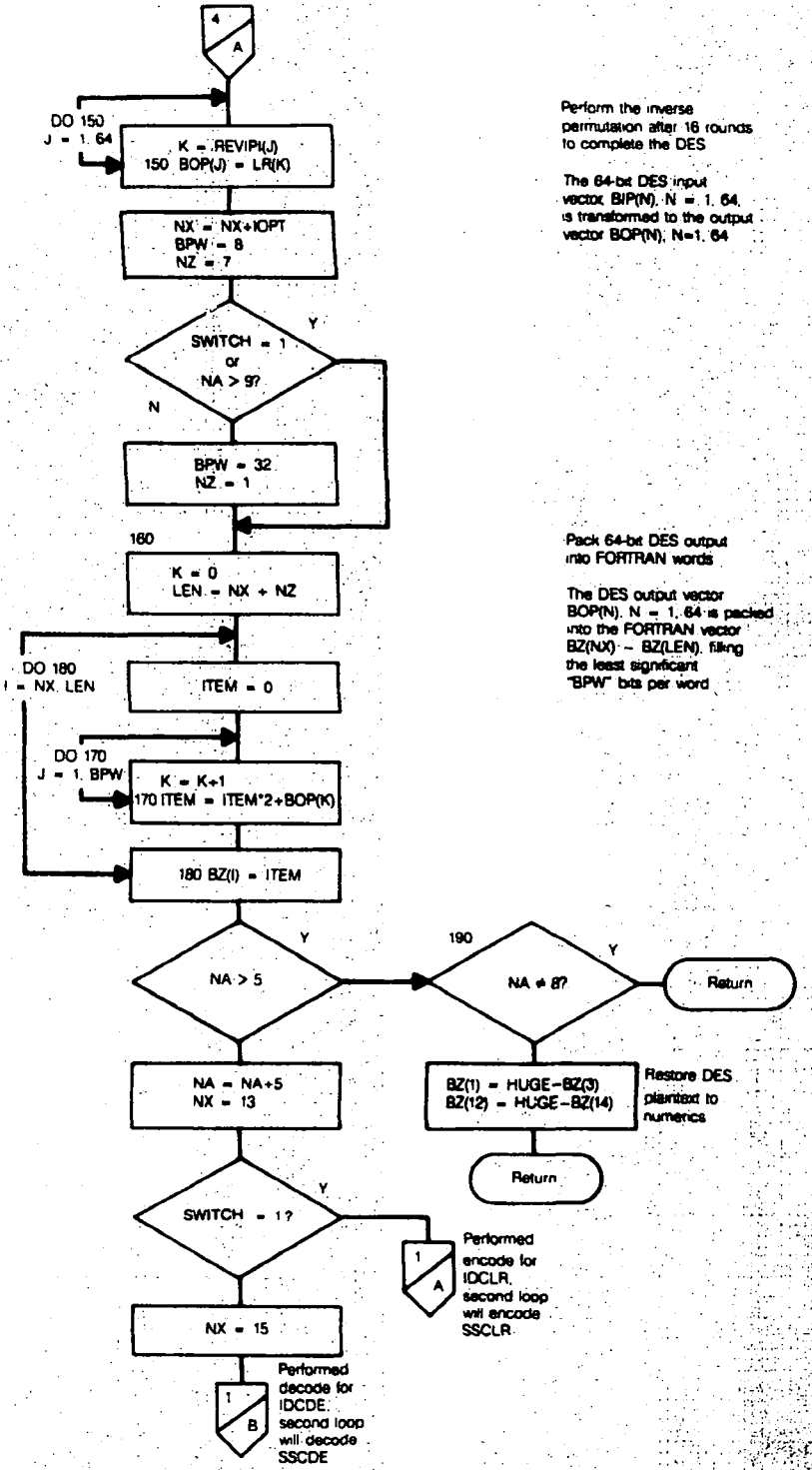
Perform substitutions using the 8 S-tables

Perform the P-permutation on the 32-bit result, perform exclusive OR of the old L and new result of the F-function

Set new L-function to be the old R-function that had been saved

APPENDIX I

Figure I.9: Subroutine DES—Segment 4



APPENDIX I

1 Relationships between the input-output variables and main program common are as follows

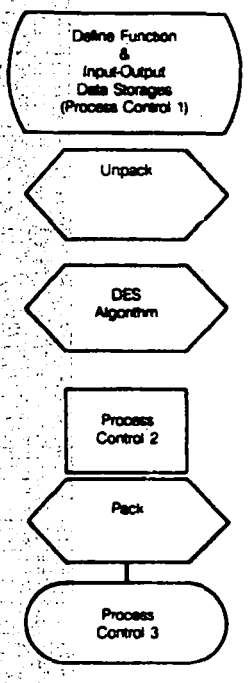
	1	4	12	15	23	31
Common	BZ(1) 3)	BZ(4) 11)	BZ(12) 14)	BZ(15) 22)	BZ(23) 30)	BZ(31) 38)
Equivalence	IDCLR	IDCDE(1) 8)	SSCLR	SSCDE(1) 8)	HEXCLR(1) 8)	HEXCDE(1) 8)

Note that the standard FORTRAN integer word [e.g. BZ(1)] occupies 4-bytes of storage, and that each byte contains 8 bits

2 Our data encryption standard (DES) algorithm performs encipher/decipher functions through the selective permutation of bits within a 64-bit binary vector. This means,

- Since FORTRAN does not provide direct reference to an individual bit, the subroutine must first "unpack" the relevant storage bytes into a 64-bit pseudo binary vector [BIP(N) N = 1, 64]
- The DES *algorithm transforms the "BIP" (binary input vector) into a 64-bit pseudo binary output vector* [BOP(1), I = 1, 64]
- The subroutine then "packs" this DES output into the BPW "least significant bits of each word" in the output storage variable

So, the subroutine might be viewed in 6 segments.



Segmt 1
Flow Chart
Figure I.6

Function SWITCH 1 is encryption, 2 decryption
NX is initial word of I/O array [e.g. BZ(NX)]
NZ is computational aid
LEN = NX+NZ is last word of array [BZ(LEN)]

Segmt 2
Flow Chart
Top Part
Figure I.7

Unpack "BPW" bits from each FORTRAN word BZ(NX) through BZ(LEN)
Create a 64-bit pseudo binary vector BIP(N), N = 1, 64

Segmt 3
Flow Chart
Middle
Figure I.7
Figure I.8
Top Figure I.9

The plaintext vector "BIP" is transformed into the 64-bit DES output vector BOP(N), N = 1, 64

Segmt 4
Top
Figure I.9

Shift program control indices to output storage, etc.
IOPT is assisting computational variable

Segmt 5
Middle
Figure I.9

The 64-bit output vector is transformed into FORTRAN words BZ(NX) through BZ(LEN), each with "BPW" bits filling the least significant positions

Segmt 6
Bottom
Figure I.9

"NA" is used as a loop index for program control
The Subroutine is translated once for a "CALL DES" and twice for "Entry" Calls

3 Preparations of numeric data for encoding

The plaintext input for USMC and SSAN is stored under numeric specification—e.g. occupying only a single word of storage. Subroutine processing fulfills two purposes.

For "Call DES(3)," the original IDCLR = BZ(1) and SSCLR = BZ(12) remain undisturbed

For the IDCLR number, two surrogate variables are created as follows

BZ(3) = HUGE−BZ(1)
BZ(2) = BZ(3)−BZ(1)

and, similarly for the SSCLR

BZ(14) = HUGE−BZ(12)
BZ(13) = BZ(14)−BZ(12)

where, HUGE = $2^{31}-1$ = 2147483647

Note that processing uses the two words [BZ(2) & BZ(3)] to provide the 64-bit vector for DES encryption of the IDCLR number—similarly with [BZ(13) & BZ(14)] for SSCLR encryption. The point is to (1) preserve original input,
(2) obtain 64-bit vector, and
(3) avoid potential of predictable "zero-strings" in high-order numeric positions.

"A consideration for abutting the two single-word variables, thus yielding a 64-bit combination, was ruled out. This would have been technically simple, but presented a difficulty in future record matches—say if either IDCLR or SSCLR had not been present (a completely different cipher).

Accordingly, for "call DES(4)" the final subroutine step reverses the process with

B(1) = HUGE−BZ(3)
B(12) = HUGE−BZ(14)

```
C*****************************************************************
C*    DES IS THE FEDERAL STANDARD (FIPS 46) FOR CRYPTOGRAPHIC    *
C*    PROTECTION OF SENSITIVE, BUT UNCLASSIFIED COMPUTER DATA.   *
C*    THE ALGORITHM IS PUBLIC KNOWLEDGE.  ENCRYPTION KEY(S) ARE  *
C*    PROTECTED BY THE USER.                                     *
C*                                                               *
C*    UTILITY PROGRAM FOR IMPLEMENTING THE DATA ENCRYPTION       *
C*    STANDARD (DES) ALGORITHM (ELECTRONIC CODEBOOK MODE).       *
C*****************************************************************
      IMPLICIT INTEGER (A-Z)
      INTEGER KEYHEX(8),KEYBIN(64),IDCDE(8),SSCDE(8),
     1 HEXCLR(8), HEXCDE(8)
      COMMON /DES1/
     1 BIP(64), BOP(64), KS(48,16), IP(64), REVIPI(64), S(64,8),
     2 E(48), P(32), LR(64), PC2(48), BX(72), BZ(46),
     3 EX(48), F(32), TEMPL(32), HUGE
C*
      EQUIVALENCE (BIP(1),KEYHEX(1)),(BOP(1),KEYBIN(1))
      EQUIVALENCE (BZ(1),IDCLR),(BZ(4),IDCDE(1))
      EQUIVALENCE (BZ(12),SSCLR),(BZ(15),SSCDE(1))
      EQUIVALENCE (BZ(23),HEXCLR(1)),(BZ(31),HEXCDE(1))
C*****************************************************************
C*   DES COMMON SHOULD NOT BE ADDRESSED WITHIN THE APPLICATION   *
C*   PROGRAM.  ADDITIONAL SPECIFICATION STATEMENTS MAY BE
C*   INSERTED IF REQUIRED                                        *
C*  1 FORMAT(64I1)
C*  2 FORMAT(8Z2)
C*   KS ARRAY IS INITIALIZED BY
C*   SUBROUTINE SETKEY(IOPT).                                    *
C*   A KEY MUST BE SPECIFIED BEFORE BEGINNING THE DES.  SETKEY   *
C*   DOES NOT HAVE TO BE CALLED AGAIN UNLESS THE USER SPECIFIES  *
C*   A KEY CHANGE. *                                             *
C*   *
C*   TWO OPTIONS ARE PROVIDED FOR KEY INPUT:                     *
C*   IOPT=1   A 64-BIT BINARY REPRESENTATION   KEYBIN 64I1       *
C*   IOPT=2   A 16-HEXADECIMAL CHARACTER STRING   KEYHEX   8Z2   *
C*
C*   SPECIFY OPTION AS IOPT=1 OR IOPT=2 ON NEXT LINE             *
      IOPT=2
C*    CHECK INPUT DEVICE OF READ STATEMENTS                      *
      IF(IOPT.EQ.2) GO TO 3
      IF(IOPT.NE.1) CALL EXIT
      READ(1,1) KEYBIN
      GO TO 4
    3 READ(1,2) KEYHEX
    4 CALL SETKEY(IOPT)


C*****************************************************************
C*   INITIALIZATION OF THE DES COMPLETED.                        *
C*   USER APPLICATION PROGRAM FOLLOWS THIS SEGMENT               *
C*****************************************************************
```

```
C*    PROGRAM TO SIMULATE DES VALIDATION                              *
C*    (E.G.,  USER APPLICATION)                                       *
C*
C*    REF: VALIDATING THE CORRECTNESS OF HARDWARE IMPLEMENTATIONS*
C*    OF THE NBS DATA ENCRYPTION STANDARD (NAT'L BUREAU OF        *
C*    STANDARDS SPECIAL PUBLICATION 500-20)                           *
C*
C*    SPECIFY TEST NUMBER AS VARIABLE ITEST                           *
      READ(1,5) ITEST
    5 FORMAT(I1)
      IF(ITEST.NE.1) GO TO 6
C-----------------------------------------------------------------
C*
C*    TEST 1 & 2:  IP & E TESTS
C*
C*    TEST 1: SET KEY=0 AND ENCRYPT THE 64-BIT DATA VECTORS E**I;
C*    I=1,2,...,64 (A SET OF BASIS VECTORS).  BASIS VECTORS HAVE
C*    ALL ZERO'S EXCEPT FOR A SINGLE 1 IN THE I-TH POSITION.
C*    COMPARE THE RESULTING CIPHER C(I) WITH THE KNOWN RESULTS
C*    (REF: P.28-29).
C*
C*    CORRECT OPERATION VERIFIES THE INITIAL PERMUTATION, IP.
C*    BECAUSE A FULL SET OF BASIS VECTORS IS PRESENTED TO THE
C*    EXPANSION MATRIX, E, THIS OPERATION IS ALSO VERIFIED.
C*
      GO TO 7
    6 IF(ITEST.NE.2) GO TO 13
    7 CONTINUE
C*
C*    TESTS 1 & 2 ARE PERFORMED SIMULTANEOUSLY IN THIS PROGRAM.
C*
C*    TEST 2: SET KEY=0 AND DECRYPT THE RESULTS C(I) OBTAINED IN
C*    TEST 1.
C*
C*    AS THE SET OF BASIS VECTORS IS RECOVERED, EACH E**I IS
C*    PRESENTED TO THE INVERSE PERMUTATION REVIPI, THUS
C*    VERIFYING IT.
C*
        DO 8 IA=1,8
        KEYHEX(I)=0
    8   HEXCLR(I)=0
        CALL HKEY(1)
        CALL SETKEY(2)
C*    KEY SET=0 AND GENERATE REQ'D PLAINTEXT INPUT
      DO 12 IA=1,8
      I=9-IA
      DO 11 JA=1,8
      J=JA-1
      HEXCLR(I)=2**J
C*    ENCRYPT THE 64-BIT VECTORS AND OUTPUT TEST 1 DATA (REF:
C*      P.28).
      CALL DES(1)
```

```
      CALL HKEY(2)
      WRITE(6,9) KEYHEX, HEXCLR, HEXCDE
    9 FORMAT(3(3X,8Z2))
C*    CLEAR HEXCLR(I) AND PROCEED IN TEST 2
      HEXCLR(I)=0
      CALL DES(2)
      WRITE(6,10) HEXCLR
   10 FORMAT('+',60X,8Z2)
   11 CONTINUE
   12 HEXCLR(I)=0
      GO TO 900
   13 IF(ITEST.NE.3) GO TO 17
C-----------------------------------------------------------------
C*
C*    TEST 3: P-OPERATOR TEST
C*
C*    TO TEST THE PERMUTATION OPERATOR, P, SET THE PLAINTEXT TO
C*    ZERO AND PROCESS THE 32 KEYS IN PTEST (REF: P.32).  THIS
C*    PRESENTS A COMPLETE SET OF BASIS VECTORS TO P.
C*
       DO 14 I=1,8
   14 HEXCLR(I)=0
   15 READ(1,16,END=900) KEYHEX
   16 FORMAT(8Z2)
      CALL HKEY(1)
      CALL SETKEY(2)
      CALL DES(1)
      CALL HKEY(2)
      WRITE(6,9) KEYHEX, HEXCLR, HEXCDE
      GO TO 15
   17 IF(ITEST.NE.4) GO TO 31
C-----------------------------------------------------------------
C*
C*    TEST 4:  PC1 & PC2 TEST
C*    PART 1: SET DATA=0 AND USE THE KEYS E**I, I=1,2,...,64,
C*    IGNORING I=8,16,...,64.  SINCE THE 56 POSSIBLE BASIS
C*    VECTORS WHICH YIELD UNIQUE KEYS ARE USED, THIS IS A
C*    COMPLETE SET OF BASIS VECTORS FOR PC1.  COMPARE THE
C*    RESULTS TO KNOWN VALUES (REF: P.30).
C*
C*    THE KEY PERMUTATION, PC1, IS VERIFIED.  SINCE THE KEY
C*    SCHEDULE CONSISTS OF LEFT SHIFTS AS I RANGES OVER THE
C*    INDEX SET, PC2 IS VERIFIED.
C*
   18 DO 19 I=1,8
   19 HEXCLR(I)=0
      READ(1,2,END=900) KEYHEX
      CALL HKEY(1)
      CALL SETKEY(2)
      CALL DES(1)
      CALL HKEY(2)
      WRITE(6,9) KEYHEX, HEXCLR, HEXCDE
C*
```

```
C*    PART 2: SET DATA=C(I) FROM PART 1 AND USE THE KEYS E**I,
C*    I=1,2,...,64, IGNORING I=8,16,...,64.  THEN DECIPHER.
C*    THIS TESTS THE RIGHT SHIFTS IN THE KEY SCHEDULE DURING
C*    DECIPHERING.
C*
       CALL DES(2)
       WRITE(6,10) HEXCLR
       GO TO 18
   31  IF(ITEST.NE.5) GO TO 50
C-------------------------------------------------------------------
C*
C*    TEST 5:  S-BOX TEST
C*
C*    SET DATA AND KEY EQUAL TO THE INPUTS DEFINED IN THE
C*    SUBSTITUTION TABLE TEST (REF: P.33).  THESE ARE A SET OF
C*    19 KEY-DATA PAIRS THAT RESULT IN EVERY ENTRY OF ALL EIGHT
C*    SUBSTITUTION TABLES BEING USED AT LEAST ONCE.  COMPARE THE
C*    RESULTS TO KNOWN VALUES.  EIGHT SUBSTITUTION TABLES WITH
C* 64-ENTRIES VERIFIED.
C*
   39  READ(1,40,END=900) KEYHEX, HEXCLR
   40  FORMAT(8Z2,3X,8Z2)
       CALL HKEY(1)
       CALL SETKEY(2)
       CALL DES(1)
       CALL HKEY(2)
       WRITE(6,9) KEYHEX, HEXCLR, HEXCDE
       GO TO 39
   50  IF(ITEST.NE.6) GO TO 900
C-------------------------------------------------------------------C*
C*    TEST 6:  DES MAINTENANCE TEST
C*    REF: MAINTENANCE TESTING FOR THE DATA ENCRYPTION STANDARD,
C*    NAT'L BUREAU OF STANDARDS SPECIAL PUBLICATION 500-61
C*
C*    THE MAINTENANCE TEST PROGRAM CREATES A CYCLING PROCESS THAT
C*    TESTS THE COMPLETE FUNCTIONALITY OF THE DES ALGORITHM AS WELL
C*    AS TESTING FOR STUCK-AT-ONE AND STUCK-AT-ZERO FAULTS AT THE
C*    VARIOUS INPUT OR OUTPUT INTERFACES (REF: P.3).  ALL FOUR
C*    MAINTENANCE TESTS HAVE THE SAME INPUT -- KEY=5555555555555555
C*    AND PLAINTEXT=FFFFFFFFFFFFFFFF.  THE INPUT NBR OF ITERATIONS
C*    DETERMINES WHICH TEST IS PERFORMED.
C*
       READ(1,501) NBR
  501  FORMAT(I2)
C*
C*    TEST 6.1:  ITERATIONS=3  --  TESTS ALL OUTPUT BITS FOR
C*    STUCK-FAULTS, AND TESTS THE P AND E MATRICES USED BY THE
C*    DES ALGORITHM.
C*    TEST 6.2:  ITERATIONS=6 -- INCLUDES TEST 1, TESTS THE
C*    S-BOXES, AND TESTS FOR STUCK-FAULTS AT ALL THE KEY AND
C*    INPUT BITS EXCEPT ONE INPUT BIT.
C*    TEST 6.3:  ITERATIONS=8   --   INCLUDES TEST 2, A COMPLETE
C*    TEST FOR STUCK-FAULTS, AND A TEST OF THE IP**(-1) MATRIX.
```

```
C*    TEST 6.4:  ITERATIONS=64  --   TESTS ALL ASPECTS OF THE
C*    ALGORITHM.
C*
C*    COMPARE RESULTS WITH KNOWN VALUES (REF: P.5).
C*
      READ(1,502) KEYHEX, HEXCLR
  502 FORMAT(8Z2)
      DO 600 I=1,NBR
      CALL HKEY(1)
      CALL SETKEY(2)
      CALL DES(1)
      CALL HKEY(5)
      CALL DES(1)
      CALL HKEY(4)
      CALL HKEY(6)
      CALL SETKEY(2)
      CALL DES(2)
      CALL HKEY(3)
      CALL HKEY(5)
  600 CONTINUE
      WRITE(6,601) KEYHEX, HEXCLR
  601 FORMAT(5X,8Z2,5X,8Z2)
  900 CONTINUE

C*    THIS IS THE END OF THE USER APPLICATION SEGMENT
C*************************************************************************
      END
      BLOCK DATA
      IMPLICIT INTEGER(A-Z)
      INTEGER S1(64), S2(64), S3(64), S4(64), S5(64), S6(64),
     1 S7(64), S8(64), PC1(56), KSFT(16)
      COMMON /DES1/
     1 BIP(64), BOP(64), KS(48,16), IP(64), REVIPI(64), S(64,8),
     2 E(48), P(32), LR(64), PC2(48), BX(72), BZ(46),
     3 EX(48), F(32), TEMPL(32), HUGE
      EQUIVALENCE (S(1,1),S1(1)), (S(1,2),S2(1))
      EQUIVALENCE (S(1,3),S3(1)), (S(1,4),S4(1))
      EQUIVALENCE (S(1,5),S5(1)), (S(1,6),S6(1))
      EQUIVALENCE (S(1,7),S7(1)), (S(1,8),S8(1))
      EQUIVALENCE (BX(1),PC1(1)),(BX(57),KSFT(1))
C*    IP IS THE INITIAL PERMUTATION FOR THE DATA.
      DATA IP/
     1 58,50,42,34,26,18,10,02,
     1 60,52,44,36,28,20,12,04,
     1 62,54,46,38,30,22,14,06,
     1 64,56,48,40,32,24,16,08,
     1 57,49,41,33,25,17,09,01,
     1 59,51,43,35,27,19,11,03,
     1 61,53,45,37,29,21,13,05,
     1 63,55,47,39,31,23,15,07/
C*    REVIPI IS THE REVERSED TABLE OF THE INITIAL PERMUTATION'S
C*    (IP'S) INVERSE.  USING THE REVERSE TABLE SAVES THE TIME
```

```
C*    OF REVERSING THE L AND R REGISTERS.
      DATA REVIPI/
     1 08,40,16,48,24,56,32,64,
     1 07,39,15,47,23,55,31,63,
     1 06,38,14,46,22,54,30,62,
     1 05,37,13,45,21,53,29,61,
     1 04,36,12,44,20,52,28,60,
     1 03,35,11,43,19,51,27,59,
     1 02,34,10,42,18,50,26,58,
     1 01,33,09,41,17,49,25,57/
C*    THE EIGHT SUBSTITUTION TABLES ARE USED IN THE DES TO
C*      SUBSTITUTE
C*    FOUR BITS FOR SIX BITS.  ONE S-TABLE IS USED FOR EACH SIX BIT
C*    TO FOUR BIT TRANSFORMATION.
      DATA S1/
     1 14,04,13,01,02,15,11,08,03,10,06,12,05,09,00,07,
     1 00,15,07,04,14,02,13,01,10,06,12,11,09,05,03,08,
     1 04,01,14,08,13,06,02,11,15,12,09,07,03,10,05,00,
     1 15,12,08,02,04,09,01,07,05,11,03,14,10,00,06,13/
      DATA S2/
     1 15,01,08,14,06,11,03,04,09,07,02,13,12,00,05,10,
     1 03,13,04,07,15,02,08,14,12,00,01,10,06,09,11,05,
     1 00,14,07,11,10,04,13,01,05,08,12,06,09,03,02,15,
     1 13,08,10,01,03,15,04,02,11,06,07,12,00,05,14,09/
      DATA S3/
     1 10,00,09,14,06,03,15,05,01,13,12,07,11,04,02,08,
     1 13,07,00,09,03,04,06,10,02,08,05,14,12,11,15,01,
     1 13,06,04,09,08,15,03,00,11,01,02,12,05,10,14,07,
     1 01,10,13,00,06,09,08,07,04,15,14,03,11,05,02,12/
      DATA S4/
     1 07,13,14,03,00,06,09,10,01,02,08,05,11,12,04,15,
     1 13,08,11,05,06,15,00,03,04,07,02,12,01,10,14,09,
     1 10,06,09,00,12,11,07,13,15,01,03,14,05,02,08,04,
     1 03,15,00,06,10,01,13,08,09,04,05,11,12,07,02,14/
      DATA S5/
     1 02,12,04,01,07,10,11,06,08,05,03,15,13,00,14,09,
     1 14,11,02,12,04,07,13,01,05,00,15,10,03,09,08,06,
     1 04,02,01,11,10,13,07,08,15,09,12,05,06,03,00,14,
     1 11,08,12,07,01,14,02,13,06,15,00,09,10,04,05,03/
      DATA S6/
     1 12,01,10,15,09,02,06,08,00,13,03,04,14,07,05,11,
     1 10,15,04,02,07,12,09,05,06,01,13,14,00,11,03,08,
     1 09,14,15,05,02,08,12,03,07,00,04,10,01,13,11,06,
     1 04,03,02,12,09,05,15,10,11,14,01,07,06,00,08,13/
      DATA S7/
     1 04,11,02,14,15,00,08,13,03,12,09,07,05,10,06,01,
     1 13,00,11,07,04,09,01,10,14,03,05,12,02,15,08,06,
     1 01,04,11,13,12,03,07,14,10,15,06,08,00,05,09,02,
     1 06,11,13,08,01,04,10,07,09,05,00,15,14,02,03,12/
      DATA S8/
     1 13,02,08,04,06,15,11,01,10,09,03,14,05,00,12,07,
     1 01,15,13,08,10,03,07,04,12,05,06,11,00,14,09,02,
```

```
     1 07,11,04,01,09,12,14,02,00,06,10,13,15,03,05,08,
     1 02,01,14,07,04,10,08,13,15,12,09,00,03,05,06,11/
C*   E IS THE EXPANSION OPERATION WHICH EXPANDS 32 BITS TO 48 BITS
      DATA E/
     1 32,01,02,03,04,05,
     1 04,05,06,07,08,09,
     1 08,09,10,11,12,13,
     1 12,13,14,15,16,17,
     1 16,17,18,19,20,21,
     1 20,21,22,23,24,25,
     1 24,25,26,27,28,29,
     1 28,29,30,31,32,01/
C*   P IS THE PERMUTATION USED IN THE MAIN DES FUNCTION.
      DATA P/
     1 16,07,20,21,
     1 29,12,28,17,
     1 01,15,23,26,
     1 05,18,31,10,
     1 02,08,24,14,
     1 32,27,03,09,
     1 19,13,30,06,
     1 22,11,04,25/
C*      PC2 IS THE PERMUTED CHOICE 2 DEFINED IN THE DES.
      DATA PC2/
     1 14,17,11,24,01,05,
     1 03,28,15,06,21,10,
     1 23,19,12,04,26,08,
     1 16,07,27,20,13,02,
     1 41,52,31,37,47,55,
     1 30,40,51,45,33,48,
     1 44,49,39,56,34,53,
     1 46,42,50,36,29,32/
C*      PC1 IS THE PERMUTED CHOICE 1 DEFINED IN THE DES.
      DATA PC1/
     1 57,49,41,33,25,17,09,
     1 01,58,50,42,34,26,18,
     1 10,02,59,51,43,35,27,
     1 19,11,03,60,52,44,36,
     1 63,55,47,39,31,23,15,
     1 07,62,54,46,38,30,22,
     1 14,06,61,53,45,37,29,
     1 21,13,05,28,20,12,04/
C*      KSFT IS THE KEY SHIFT VECTOR DEFINED IN THE DES.
      DATA KSFT/1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1/
C*   HUGE IS A CONSTANT USED IN THE PREPARATION OF NUMERIC DATA
C*   FOR ENCODING.  ITS PURPOSE IS TO AVOID THE POTENTIAL OF
C*   PREDICTABLE 'ZERO-STRINGS' ON HIGH ORDER NUMERIC
C*    POSITIONS.  HUGE = 2**31-1.
      DATA HUGE/2147483647/
      END

C*   SUBROUTINE TO FACILITATE VALIDATION TESTING.  DOES NOT HAVE
```

```
C*    ANY USE IN GENERAL ENCIPHER/DECIPHER PROGRAM.  PROGRAM STORES
C*    KEY SO THAT IT CAN BE RECALLED FOR ITERATIVE TESTING.
       SUBROUTINE HKEY(ITEM)
       IMPLICIT INTEGER(A-Z)
       COMMON /DES1/
      1 BIP(64), BOP(64), KS(48,16), IP(64), REVIPI(64), S(64,8),
      2 E(48), P(32), LR(64), PC2(48), BX(72), BZ(46),
      3 EX(48), F(32), TEMPL(32), HUGE
       GO TO (1,3,5,7,9,11), ITEM
       CALL EXIT
C*     ITEM=1    SAVE SETKEY VALUE IN BZ(39) - BZ(46)
      1 DO 2 I=1,8
        J=I+38
      2 BZ(J)=BIP(I)
        RETURN
C*     ITEM=2    RETURN BZ(39) - BZ(46) VALUE TO KEYHEX
      3 DO 4 I=1,8
        J=I+38
      4 BIP(I)=BZ(J)
        RETURN
C*     ITEM=3    MOVE HEXCLR VALUE INTO KEYHEX AND SAVE AREA
      5 DO 6 I=1,8
        J=I+38
        K=I+22
        BIP(I)=BZ(K)
      6 BZ(J)=BZ(K)
        RETURN
C*     ITEM=4    MOVE HEXCDE VALUE INTO KEYHEX AND SAVE AREA
      7 DO 8 I=1,8
        J=I+38
        K=I+30
        BIP(I)=BZ(K)
      8 BZ(J)=BZ(K)
        RETURN
C*     ITEM=5    MOVE HEXCDE VALUE INTO HEXCLR POSITION
      9 DO 10 I=23,30
        J=I+8
     10 BZ(I)=BZ(J)
        RETURN
C*     ITEM=6    MOVE HEXCLR VALUE INTO HEXCDE STORAGE
     11 DO 12 I=23,30
        J=I+8
        RETURN
        END

C*    SUBROUTINE TO PERFORM DES KEY SCHEDULE.
       SUBROUTINE SETKEY(IOPT)
       IMPLICIT INTEGER(A-Z)
       INTEGER CD(64), C(28), D(36), PC1(56), KSFT(16)
       COMMON /DES1/
      1 BIP(64), BOP(64), KS(48,16), IP(64), REVIPI(64), S(64,8),
      2 E(48), P(32), LR(64), PC2(48), BX(72), BZ(46),
```

```
     3 EX(48), F(32), TEMPL(32), HUGE
       EQUIVALENCE (LR(1),CD(1),C(1)),(LR(29),D(1))
       EQUIVALENCE (BX(1),PC1(1)),(BX(57),KSFT(1))
C*    IOPT=1 SPECIFIES BINARY KEY; IOPT=2 SPECIFIES A HEXADECIMAL
C*      KEY
       IF(IOPT.EQ.1) GO TO 7
       IF(IOPT.NE.2) CALL EXIT
       LEN=8
       BPW=8
     5 K=0
C*    THIS LOOP UNPACKS A HEX-STRING OF 8 WORDS AT 8 BITS-PER-WORD
C*    OF SIGNIFICANT INFORMATION (LEAST SIGNIFICANT BITS) INTO A
C*    64-BIT PSEUDO BINARY VECTOR.
       DO 6 I=1,LEN
       ITEM=BIP(I)
       DO 6 J=1,BPW
       IEX=2**(BPW-J)
       IBIT=ITEM/IEX
       K=K+1
       BOP(K)=IBIT
     6 ITEM=ITEM-IBIT*IEX
C*    LOOP WHICH PERFORMS THE FIRST PERMUTATION ON THE KEY,
C*      REMOVING
C*    THE PARITY BITS AND SETTING UP THE C AND D REGISTERS.
     7 DO 100 I=1,56
       J=PC1(I)
   100 CD(I)=BOP(J)
C*    LOOP WHICH COMPUTES THE KS ARRAY FOR THE DES.  THIS RUNS FAST
C*    THE KS NEED ONLY BE COMPUTED ONCE FOR ALL THE ENCRYPT AND
C*    DECRYPT OPERATIONS WHICH USE THIS KEY.
       DO 400 I=1,16
       IEX=KSFT(I)
       DO 300 J=1,IEX
       LEN=C(1)
       BPW=D(1)
       DO 200 K=1,27
       C(K)=C(K+1)
   200 D(K)=D(K+1)
       C(28)=LEN
   300 D(28)=BPW
       DO 400 J=1,48
       K=PC2(J)
   400 KS(J,I)=CD(K)
       RETURN
       END
C*    THIS SUBROUTINE PROVIDES FOR THE ENCRYPTION/DECRYPTION OF
C*    MAIN PROGRAM VARIABLES.  SWITCH=1 IS ENCRYPTION, SWITCH=2 IS
C*    DECRYPTION MODE FOR THE HEXADECIMAL VARIABLES.  SWITCH=3 AND
C*    SWITCH=4 ARE USED FOR SPECIAL FORMAT NUMERIC VARIABLES.
       SUBROUTINE DES(SWITCH)
       IMPLICIT INTEGER(A-Z)
       INTEGER L(32),R(32)
       COMMON /DES1/
      1 BIP(64), BOP(64), KS(48,16), IP(64), REVIPI(64), S(64,8),
```

```
     2 E(48), P(32), LR(64), PC2(48), BX(72), BZ(46),
     3 EX(48), F(32), TEMPL(32), HUGE
      EQUIVALENCE (LR(1),L(1)), (LR(33),R(1))
C*   THIS SEGMENT OF THE SUBROUTINE DEFINES THE FUNCTION AND INPUT
C*   STORAGE LOCATIONS.
      GO TO (1,1,10,30),SWITCH
    1 NA=10
      NX=23
      IOPT=8
      IF(SWITCH.EQ.1) GO TO 40
      NX=31
      IOPT=-8
      IF(SWITCH.EQ.2) GO TO 40
      CALL EXIT
C*
C*   CALL DES(3) GENERATES CIPHERTEXT 'IDCDE' AND 'SSCDE' CORRES-
C*   PONDING TO THE MAIN PROGRAM'S 'IDCLR' AND 'SSCLR' VALUES.
   10 SWITCH=1
      NA=1
      NX=2
      IOPT=2
C*
C*   THIS SECTION STORES THE PLAINTEXT INPUT FOR IDCLR AND SSCLR
C*     UNDER
C*   NUMERIC SPECIFICATION--E.G., OCCUPYING ONLY A SINGLE WORD OF
C*   STORAGE.  HUGE IS USED TO SPECIFY THE SECOND OF THE TWO WORDS
C*   WHICH, TOGETHER, PROVIDE THE 64-BIT VECTOR FOR DES
C*   ENCRYPTION.
      BZ(3)=HUGE-BZ(1)
      BZ(2)=BZ(3)-BZ(1)
      BZ(14)=HUGE-BZ(12)
      BZ(13)=BZ(14)-BZ(12)
C*
   20 NZ=1
      BPW=32
      GO TO 50
C*   CALL DES(4) GENERATES PLAINTEXT 'IDCLR' AND 'SSCLR' C*
       CORRESPONDING
C*   TO THE MAIN PROGRAM'S 'IDCDE' AND 'SSCDE' CIPHERTEXT.
   30 SWITCH=2
      NA=3
      NX=4
      IOPT=-2
   40 NZ=7
      BPW=8
C*
C*   THIS LOOP UNPACKS IDCLR, SSCLR, OR HEXCLR INTO A 64-BIT
C*     PSEUDO
C*   BINARY VECTOR.
   50 K=0
      LEN=NX+NZ
      DO 60 I=NX,LEN
      ITEM=BZ(I)
      DO 60 J=1,BPW
```

```
      IEX=2**(BPW-J)
      IBIT=ITEM/IEX
      K=K+1
      BIP(K)=IBIT
   60 ITEM=ITEM-IBIT*IEX
C*    THIS IS THE BEGINNING OF THE DES ALGORITHM.
      N=0
      IEX=1
      IF(SWITCH.EQ.1) GO TO 70
      N=17
      IEX=-1
C*    THIS LOOP PERFORMS THE INITIAL PERMUTATION OF THE INPUT.
   70 DO 80 I=1,64
      K=IP(I)
   80 LR(I)=BIP(K)
      DO 140 IBIT=1,16
C*
C*    THIS LOOP WILL PRODUCE OUTPUT SHOWING THE PROGRESSION OF THE
C*    64-BIT PSEUDO BINARY VECTOR THROUGH THE 16 S-BOXES DURING
C*    ENCRYPTION/DECRYPTION.  SAMPLE RUN AND OUTPUT IS INCLUDED
C*    IN THE TEST RUNS SECTION.
C*
C*    BEGINNING OF THE MAIN LOOP WHICH ENCRYPTS/DECRYPTS FOR 16
C*      RNDS.
      N=N+IEX
C*      LOOP WHICH SAVES THE R-REGISTER.
      DO 100 I=1,32
  100 TEMPL(I)=R(I)
C*    LOOP WHICH EXPANDS THE R-REGISTER USING THE E-FUNCTION AND
C*    DOES THE XOR OF THE KEY SCHEDULE SUBKEY AND THE EXPANDED
C*      R.
      DO 110 I=1,48
      EX(I)=1
      K=E(I)
      IF(R(K) .EQ. KS(I,N)) EX(I)=0
  110 CONTINUE
C*    LOOP WHICH DOES THE SUBSTITUTIONS USING THE 8 S-TABLES.
      DO 120 IA=1,8
      J=IA-1
      K=J*6+1
      ITEM=EX(K)*32+EX(K+5)*16+EX(K+1)*8+EX(K+2)*4+EX(K+3)*2
      1+EX(K+4)
      SUB=S(ITEM+1,J+1)
      K=J*4
      F(K+1)=SUB/8
      F(K+2)=MOD(SUB,8)/4
      F(K+3)=MOD(SUB,4)/2
  120 F(K+4)=MOD(SUB,2)
C*    LOOP WHICH DOES THE P PERMUTATION OPERATION TO THE 32-BIT
C*    RESULT AND ALSO DOES THE XOR OF THE OLD L AND THE NEW RESULT
C*    OF THE F FUNCTION.
      DO 130 J=1,32
```

```
      R(J)=1
      K=P(J)
      IF(L(J) .EQ. F(K)) R(J)=0
  130 CONTINUE
C*    SETS THE NEW L TO BE THE OLD R THAT HAD BEEN SAVED.
      DO 140 J=1,32
  140 L(J)=TEMPL(J)
C*    LOOP PERFORMS THE INVERSE PERMUTATION AFTER THE 16 ROUNDS TO
C*    COMPLETE THE DES.
      DO 150 J=1,64
      K=REVIPI(J)
  150 BOP(J)=LR(K)
C*
C*    SECTION INDEXES LOCAL VARIABLES TO DEFINE DES OUTPUT
C*    STORAGE.
      NX=NX+IOPT
      BPW=8
      NZ=7
      IF(SWITCH.EQ.1) GO TO 160
      IF(NA.GT.9) GO TO 160
      BPW=32
      NZ=1
C*    DES OUTPUT VECTOR IS "PACKED" INTO THE FORTRAN VECTOR BZ(NX)
C*    THROUGH BZ(LEN), FILLING THE LEAST SIGNIFICANT "BPW" BITS OF
C*    EACH WORD.
  160 K=0
      LEN=NX+NZ
      DO 180 I=NX,LEN
      ITEM=0
      DO 170 J=1,BPW
      K=K+1
  170 ITEM=ITEM*2+BOP(K)
  180 BZ(I)=ITEM
C*    END OF SUBROUTINE LOOPS
      IF(NA.GT.5) GO TO 190
      NA=NA+5
      NX=13
C*    IF FIRST LOOP PERFORMED ENCODE FOR 'IDCLR', SECOND LOOP WILL
C*    ENCODE SSCLR.  IF FIRST LOOP PERFORMED DECODE OF IDCDE, THE
C*    SECOND LOOP WILL DECODE SSCDE.
      IF(SWITCH.EQ.1) GO TO 20
      NX=15
      GO TO 40
  190 IF(NA.NE.8) RETURN
C*      THIS SECTION RESTORES DES PLAINTEXT.
      BZ(1)=HUGE-BZ(3)
      BZ(12)=HUGE-BZ(14)
      RETURN
      END
//STEP2         EXEC  FORVLKGO
//GO.FT01F001   DD    *
0000000000000000
```

APPENDIX I

```
6
64
5555555555555555
FFFFFFFFFFFFFFFF
/*
      246E9DB9C550381A      EC6AF9EE48717817
```

EXAMPLES OF USER APPLICATION PROGRAMS

The following user application modules would be inserted in the DES Utility Main in the place of the DES validation module. The result of this substitution would be a customized FORTRAN program to perform encryption and decryption in accordance with the DES algorithm.

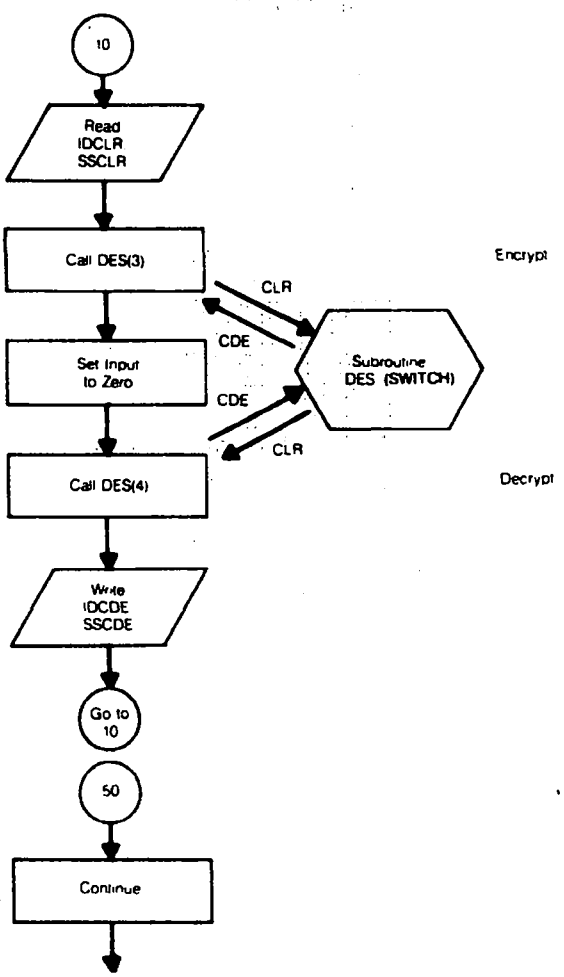
TWO ID NUMBERS

This example is set up for encrypting 2 identification numbers. The first is assumed to have 7 digits (or, at least is located in a 7-character field), the second to have 9 digits. These particular values were the ones used in GAO's Marine Corps retirement study. The first number was a military ID and the second was a Social Security number (treated as a 9-digit integer). The logic of the program is such that the first number need not be present. However, the read statement must be modified accordingly and an (unchanging) initial value should be set.

The program logic is illustrated in figure II.1 on page 56. The FORTRAN code is as follows:

```
   10 READ(1,91,END=50) IDCLR, SSCLR
C*      IDCLR IS THE 7-DIGIT ID, SSCLR IS A 9-DIGIT SOCIAL
C*      SECURITY ACCOUNT NUMBER
C*      THE NEXT TWO LINES ARE ONLY TO ECHO CHECK THE INPUT
      WRITE(6,92) IDCLR, SSCLR
      CALL DES(3)
C*      THIS WILL ENCRYPT THE VARIABLES IDCLR AND SSCLR, AND
C*      PLACE THE ENCIPHERED VALUES IN IDCDE AND SSCDE.
C*      THE NEXT TWO LINES PRINT THE ENCIPHERED VALUES.  NOTE
C*      THE USE OF HEXADECIMAL FORMAT.
      WRITE(6,93) IDCDE, SSCDE
C*      DESTROY INPUT TO DEMONSTRATE RECOVERY FROM CODE
      IDCLR=0
      SSCLR=0
      CALL DES(4)
C*      DEMONSTRATE RECOVERY OF CLEARTEXT
      WRITE(6,94) IDCLR, SSCLR
      GO TO 10
   91 FORMAT(I7,I9)
   92 FORMAT(1H ,/,I7,3X,I9)
   93 FORMAT(1H+,22X,8Z2,3X,8Z2)
   94 FORMAT(1H ,  17,3X,I9)
   50 CONTINUE
C*      LOCATION OF ENDING ROUTINE
```

Figure II.1: Processing Two ID Numbers



HEXADECIMAL INPUT AND OUTPUT STRINGS

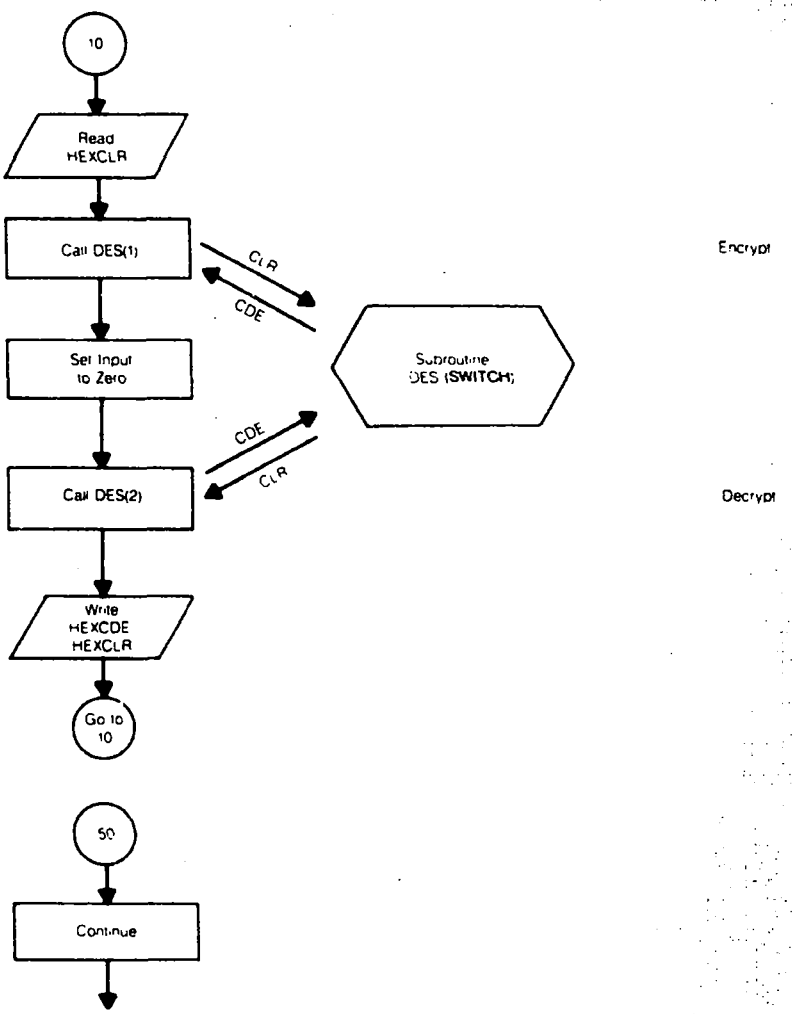
This example assumes that the characters to be encrypted are a 16-digit hexadecimal string. As in the previous example, the input will be echoed, enciphered, set to zero, and recreated from the deciphering process. The program is set to cycle until the data are exhausted.

The program logic is illustrated in figure II.2 The FORTRAN code is as follows:

```
C*      NOTE THAT HEXCLR IS AN 8-POSITION INTEGER VECTOR
   10 READ(1,91,END=50) HEXCLR
C*       ECHO CHECK OF INPUT
      WRITE(6,92) HEXCLR
C*       PERFORM THE ENCRYPTION AND COMPUTE HEXCDE
      CALL DES(1)
```

```
C*      DESTROY INPUT TO DEMONSTRATE RECOVERY FROM HEXCDE
      DO 8 I=1,8
    8 HEXCLR(I)=0
C*      PERFORM THE DECRYPTION AND COMPUTE HEXCLR
      CALL DES (2)
      WRITE(6,93) HEXCDE, HEXCLR
      GO TO 10
   91 FORMAT(8Z2)
   92 FORMAT(1H ,8Z2)
   93 FORMAT(1H+,19X,8Z2,3X,8Z2)
   50 CONTINUE
C*      LOCATION OF ENDING ROUTINE
      END
```

Figure II.2: Processing Hexadecimal Numbers



ALPHANUMERIC STRINGS

This example would be used to encipher alphanumeric character strings--that is, normal text. The method used is essentially the same as the preceding example. Instead of reading two hexadecimal characters into an integer variable, this code will read one alphanumeric character into an integer variable.

The program flowchart is illustrated in figure II.3. The FORTRAN code is as follows:

```
C*      NOTE THAT HEXCLR IS STILL USED TO HOLD THE INPUT
   10 READ(1,91,END=50) HEXCLR
C*      ECHO CHECK OF INPUT
      WRITE(6,92) HEXCLR
C*      RIGHT SHIFT 24 BITS INTO HEXADECIMAL POSITION
      DO 15 I=1,8
   15 HEXCLR(I)=ISHFT(HEXCLR(I),-24)
C*      PERFORM THE ENCRYPTION AND COMPUTE HEXCDE
      CALL DES(1)
C*      DESTROY INPUT TO DEMONSTRATE RECOVERY FROM HEXCDE
      DO 20 I=1,8
   20 HEXCLR(I)=0
C*      PERFORM THE DECRYPTION AND COMPUTE HEXCLR
      CALL DES(2)
      WRITE(6,93) HEXCDE, HEXCLR
C*      LEFT SHIFT 24 BITS INTO CHARACTER POSITION
      DO 25 I=1,8
   25 HEXCLR(I)=ISHFT(HEXCLR(I),24)
C*      DEMONSTRATE RECOVERY OF CLEARTEXT
      WRITE(6,94) HEXCLR
      GO TO 10
   91 FORMAT(8A1)
   92 FORMAT(1H ,8A1)
   93 FORMAT(1H+,11X,8Z2,3X,8Z2)
   94 FORMAT(1H+,49X,8A1)
C*      LOCATION OF ENDING ROUTINE
   50 CONTINUE
      END
```

The FORTRAN function ISHFT is used to shift the internal representation of the character being manipulated between the two formats. When the FORTRAN format type "A" is used to read character data into an integer variable, the characters are read from left to right and stored from left to right in the appropriate memory location. Subroutine DES expects to find the characters represented as numbers and stored in the rightmost portion of the computer word, as they are when read by a "Z" format type (hexadecimal). This is illustrated by these memory representations of the two storage formats for the character "a":

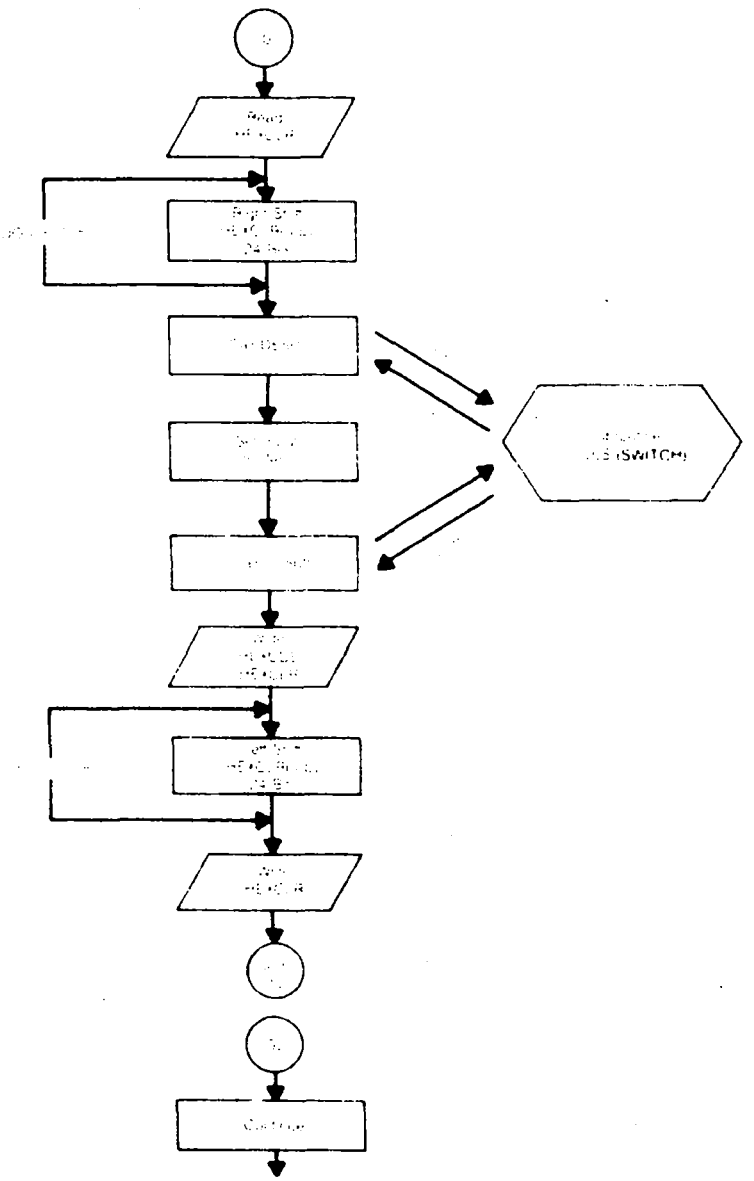
```
a   -   -   -                    -   -   -   a

character format                 hexadecimal format
```

Function ISHFT shifts the cleartext character 24 places to the right before encryption. It is used again after decryption to shift the recovered cleartext back to the left of the computer word so that it may be output as a character.

The encrypted character is not shifted because not all of the resulting patterns are printable characters. In this case, it is reasonable to leave them in their hexadecimal numeric representation and, if necessary, view and print them as hexadecimal numbers.

Figure II.3: Processing Alphanumeric Characters



APPENDIX II

NUMERIC ID WITHIN RECORD

It is assumed in this case that there is a numeric identifier (possibly a 9-digit Social Security number) within a larger record of interest. The identifier is to be encrypted and all other data in the record is to be written without modification. Specific assumptions are

	Input	Output
Record length	100	107
ID length	9	16
ID location in record	10-18	10-25

A set of IBM FORTRAN IV commands to accomplish this are illustrated below. The input records are read from logical unit 12 and the output records are written to logical unit 13. Note that the unchanged data are read and written into integer variables using an "A" format. This subterfuge is not necessary in a language that supports direct character input and output.

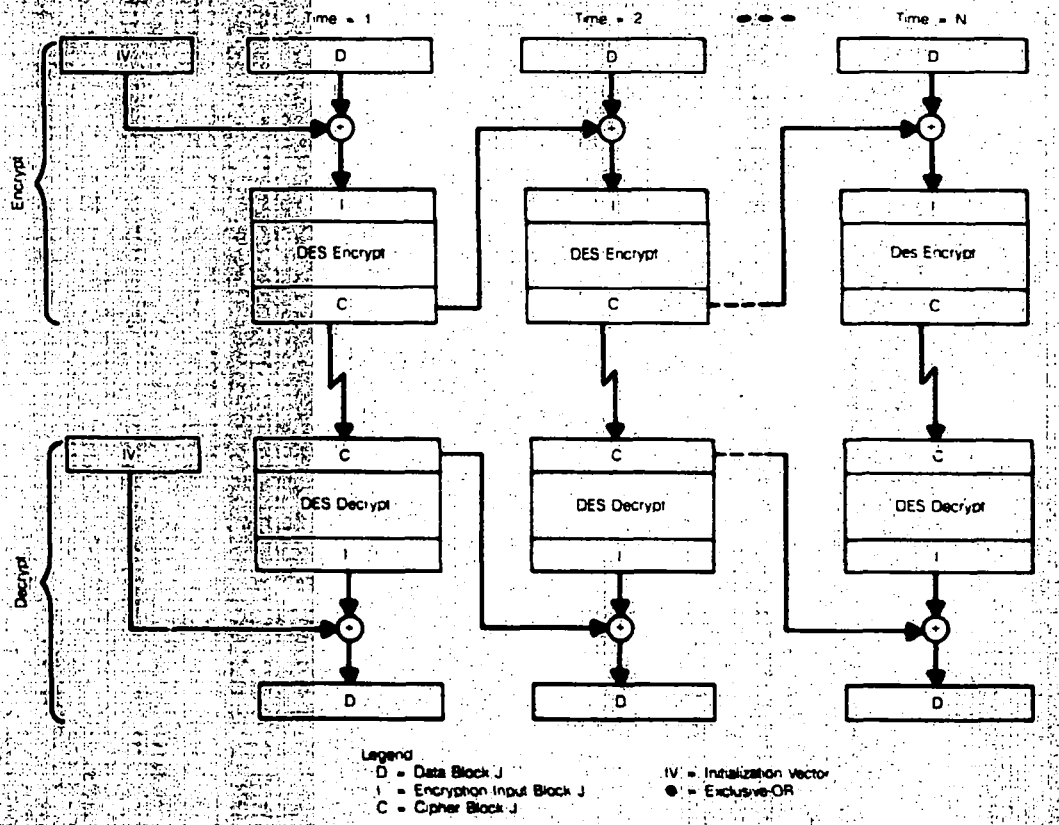
```
      INTEGER BLOCK1(3), BLOCK2(21)
      IDCLR=0
            ...
   10 READ(12,91,END=50) BLOCK1, SSCLR, BLOCK2
      CALL DES(3)
      WRITE(13,92) BLOCK1, SSCDE, BLOCK2
      GO TO 10
   91 FORMAT(2A4,A1,I9,20A4,A2)
   92 FORMAT(2A4,A1,8Z2,20A4,A2)
   50 CONTINUE
            ...
```

DES PROGRAM IN C LANGUAGE

This appendix documents a software implementation of the DES algorithm written in the C language. It has been tested on IBM-PC microcomputers using a MICROSOFT C compiler. It is a more recent development than the FORTRAN program listed in appendix I and is a much faster-running program than the FORTRAN implementation. The greater speed of this implementation makes this program a better candidate for use when the greater security of the cipher block chaining mode is desired.[1]

Briefly, the CBC mode is defined as follows. (See figure III.1 for a schematic view of the CBC mode.) As message to be encrypted is first divided into 64-bit blocks. In most applications this will be done by segmentng the message to be

Figure III.1: Cipher Block Chaining Mode



[1]Note that the program is configured to operate in the electronic codebook mode. The extra steps necessry for cipher block chaining are performed outside the DES algorithm proper.

encrypted into consecutive 8-character groups. If a microcomputer file were to be encrypted, the 64-bit blocks would be taken as consecutive groups of 8 bytes each. The last block would be padded as necessary to make it 8 bytes (or characters). In addition to the key required for the DES algorithm, the CBC mode also requires use of initialization vector (IV).

In CBC encryption, the first plaintext input block is exclusive-ORed with the IV. The resulting block is then input into the DES encryption algorithm adn the output is the first ciphertext block. Next, this ciphertext block is exclusive-ORed with the second plaintext block and the result is fed into the DES algorithm. This output is the second ciphertext block. This plaintext block is repeated until the end of the message is reached, hence the name.

CBC decryption is similar. The first ciphertext block is passed through the DES algorithm resulting in a "decrypted" output block. This block is exclusive-ORed with the original IV and produces the first plaintext block. The second ciphertext block is decrypted and the output is exclusive-ORed with the first ciphertext block to produce the second plaintext block, and so on.

The technical sophistication to use the DES in the CBC mode is obviously somewhat greater than that required for using the electronic codebook mode. However, this added complexity is balanced by the greater security of the CBC mode and its use in message authentication. More detailed information on the specifics of implementing the CBC mode may be found on pp. 5-8 of DES Modes of Operation (FIPS 81) and pp. 16-18, and 25 of Guidelines for Implementing and Using the NBS Data Encryption Standard (FIPS 74).

```
/*****************************************************************
 *                                                               *
 *  THIS PROGRAM IMPLEMENTS THE FEDERAL INFORMATION PROCESSING   *
 *  DATA ENCRYPTION STANDARD (FIPS PUB 46 - 1977 January 15).    *
 *                                                               *
 *  Original author: Ken Thompson                                *
 *  Modified by: Brian Lucas                                     *
 *               Burt S. Kaliski, 1983 June 16                   *
 *               David M. Balenson, November 1984.               *
 *                                                               *
 *  VERSION 2.00                                                 *
 *                                                               *
 *  NOTE: Produced at the National Bureau of Standards (NBS).    *
 *        Not supported by NBS and not subject to copyright.     *
 *                                                               *
 *****************************************************************/

/*****************************************************************
 *  SETKEY()                                                     *
 *****************************************************************/

#define FALSE  0
#define TRUE   1
#define CHECK  1
#define GENERATE 2
#define SINGLE 1
#define PAIR   2


/* Permuted-Choice 1 (PC1) from the key bits to yield C and D.
 * Note: bits 8,16,24,32,40,48,56,and 64 are left out since
 * they are intended for a parity check.
 */
char    PC1_C[] = {
        57,49,41,33,25,17, 9,
         1,58,50,42,34,26,18,
        10, 2,59,51,43,35,27,
        19,11, 3,60,52,44,36,
};

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

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

/* Permuted-Choice 2 (PC2) to pick out the 48-bit subkeys
 * from the CD array.
 */
char    PC2_C[] = {
        14,17,11,24, 1, 5,
         3,28,15, 6,21,10,
        23,19,12, 4,26, 8,
        16, 7,27,20,13, 2,
};

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

/* The key schedule of 16 48-bit subkeys generated from
 * the key.
 */
char    KS[16][48];

/* Set up the key schedule from the key. */
setkey(sw1,sw2,pkey)
int sw1;                /* parity: 0=ignore,1=check,2=generate */
int sw2;                /* type cryption: 0=encrypt,1=decrypt */
char *pkey;             /* 64-bit key packed into 8 bytes */
{
        register i, j, k;
        int ii,t;
        /* Following arrays are declared static to speed up
         * indexing.  The C and D arrays used to calculate the
         * key schedule.
         */
        static char C[28], D[28];
        /* The key array is a one bit per byte version of pkey */
        static char key[64];


        /* Check 'parity' parameter */
        if ((sw1 != 0) && (sw1 != 1) && (sw1 != 2)) {
                printf("\007*** setkey: bad parity",
                        " parameter ***\n");
                return(FALSE);
        }

        /* Check 'type of cryption' parameter */
        if ((sw2 != 0) && (sw2 != 1)) {
                printf("\007*** setkey: bad encrypt/",
                        "decrypt parameter ***\n");
```

```
                return(FALSE);
        }

        /* Check for ODD Key Parity */
        if ((sw1 == 1) && (!check_parity(CHECK,pkey,SINGLE)))
                return(FALSE);

        /* Generate ODD Key Parity */
        if (sw1 == 2)
                check_parity(GENERATE,pkey,SINGLE);

        /* Unpack KEY so only one bit per byte */
        unpack(8,8,pkey,key);

        /* Generate C and D by permuting the key with PC1.
         * The low order bit of each 8-bit char is not used,
         * so C and D are only 28 bits apiece.                    */
        for (i=0; i<28; i++) {
                C[i] = key[PC1_C[i]-1];
                D[i] = key[PC1_D[i]-1];
        }


        /*  To generate Ki, rotate C and D according to schedule
         *  and pick up a permutation using PC2.                   */
        for (i=0; i<16; i++) {
                /* Set direction for encrypt/decrypt */
                ii = sw2 ? 15-i : i;
                /* Rotate. */
                for (k=0; k<shifts[i]; k++) {
                        t = C[0];
                        for (j=0; j<28-1; j++)
                                C[j] = C[j+1];
                        C[27] = t;
                        t = D[0];
                        for (j=0; j<28-1; j++)
                                D[j] = D[j+1];
                        D[27] = t;
                }
                /* Get Ki. Note C and D are concatenated. */
                for (j=0; j<24; j++) {
                        KS[ii][j] = C[PC2_C[j]-1];
                        KS[ii][j+24] = D[PC2_D[j]-28-1];
                }
        }
        return(TRUE);
}
/*****************************************************************
 *  DES()                                                        *
 *****************************************************************/

/* Initial permutation, IP */
char    IP[] = {
        58,50,42,34,26,18,10, 2,
        60,52,44,36,28,20,12, 4,
```

```
        62,54,46,38,30,22,14, 6,
        64,56,48,40,32,24,16, 8,
        57,49,41,33,25,17, 9, 1,
        59,51,43,35,27,19,11, 3,
        61,53,45,37,29,21,13, 5,
        63,55,47,39,31,23,15, 7,
};

/* Final permutation, FP = IP^(-1)  [NOT USED] */
char    FP[] = {
        40, 8,48,16,56,24,64,32,
        39, 7,47,15,55,23,63,31,
        38, 6,46,14,54,22,62,30,
        37, 5,45,13,53,21,61,29,
        36, 4,44,12,52,20,60,28,
        35, 3,43,11,51,19,59,27,
        34, 2,42,10,50,18,58,26,
        33, 1,41, 9,49,17,57,25,
};

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

```
/* The E bit-selection table. */
char    E[] = {
        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,
};

/* P is a permutation on the selected combination of current L
 * and key.
 */
char    P[] = {
        16, 7,20,21,
        29,12,28,17,
         1,15,23,26,
         5,18,31,10,
         2, 8,24,14,
        32,27, 3, 9,
        19,13,30, 6,
        22,11, 4,25,
};

/* The 8 selection functions.  For some reason, they are
 * 0-origin index, unlike everything else.
 */
char    S[8][64] = {
        14, 4,13, 1, 2,15,11, 8, 3,10, 6,12, 5, 9, 0, 7,
         0,15, 7, 4,14, 2,13, 1,10, 6,12,11, 9, 5, 3, 8,
         4, 1,14, 8,13, 6, 2,11,15,12, 9, 7, 3,10, 5, 0,
        15,12, 8, 2, 4, 9, 1, 7, 5,11, 3,14,10, 0, 6,13,

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

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

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

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

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

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

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

APPENDIX III

```
/************************************
 * ENCRYPT/DECRYPT 64-BIT BLOCK. *
 ************************************/

des(in,out)
char *in;               /* packed 64-bit INPUT block */
char *out;              /* packed 64-bit OUTPUT block */
{
        int ii;
        register t, j, k;
        /* Following arrays are declared static to speed up
         * indexing.  The current block, one bit per byte version
         */
        static char block[64];
        /* The 2 halves of the current block, L and R must be
         * contiguous
         */
        static char L[32], R[32], tempL[32], f[32], tempf[32];
        /* The combination of the key and input, before
         * selection.
         */
        static char preS[48];

        /* Unpack the INPUT block */
        unpack(8,8,in,block);

        /* First, permute the bits in the input */
        for (j=0; j<64; j++)
                L[j] = block[IP[j]-1];
        /* Perform an encryption operation 16 times. */
        for (ii=0; ii<16; ii++) {
                /* Save the R array, which will be the new L. */
                for (j=0; j<32; j++)
                        tempL[j] = R[j];
                /* Expand R to 48 bits using the E selector;
                 * exclusive-or with the current key bits.
                 */
                for (j=0; j<48; j++)
                        preS[j] = R[E[j]-1] ^ KS[ii][j];
```

```
                /* The pre-select bits are now considered in 8
                 * groups of 6 bits each.  The 8 selection
                 * functions map these 6-bit quantities into 4-bit
                 * quantities and the results permuted to make an
                 * f(R, K).  The indexing into the selection
                 * functions is peculiar.
                 */
                for (j=0; j<8; j++) {
                        k = 6*j;
                        t = preS[k+0]; t <<= 1;
                        t |= preS[k+5]; t <<= 1;
                        t |= preS[k+1]; t <<= 1;
                        t |= preS[k+2]; t <<= 1;
                        t |= preS[k+3]; t <<= 1;
                        t |= preS[k+4];
                        t = S[j][t];
                        k = 4*j;
                        f[k+3] = t&01; t >>= 1;
                        f[k+2] = t&01; t >>= 1;
                        f[k+1] = t&01; t >>= 1;
                        f[k+0] = t&01;
                }
                /* The new R is L ^ f(R, K).
                 * The f here has to be permuted first, though.
                 */
                for (j=0; j<32; j++)
                        R[j] = L[j] ^ f[P[j]-1];
                /* Finally, the new L (the original R) is copied
                * back.
                 */
                for (j=0; j<32; j++)
                        L[j] = tempL[j];
        }

        /* Final output is reverse inverse permutation. */
        for (j=0; j<64; j++)
                block[j] = L[RFP[j]-1];

        /* Pack data into 8 bits per byte
        for (j=0; j<8; j++) {
                t = 0;
                for (k=0; k<8; k++)
                        t = (t<<1) + block[8*j+k];
                out[j] = t;
        } */
        pack(8,8,out,block);
}
```

```
/*****************************************************************
 *  CHECK_PARITY()  Function to check &/or generate odd parity   *
 *  for the bytes of a key. Returns TRUE if parity is ODD,       *
 *  FALSE otherwise.                                             *
 *****************************************************************/
check_parity(sw,pkey,len)
int sw;          /* parity: 1=check&report, 2=generate */
char *pkey;      /* ptr to packed 64-bit KEY (8 bytes) */
int len;         /* length: 1=single, 2=double */
{
        register i,j,k;
        int parity;
        char key[128];
        char *sptr = pkey;

        if ((sw != 1) && (sw != 2)) {
                printf("\007*** check_parity: bad",
                        " parity parameter ***\n");
                sw = 1;
        }

        if ((len != 1) && (len != 2)) {
                printf("\007*** check_parity: bad",
                        " key length ***\n");
                len = 1;
        }

        /* Unpack Key */
        unpack(len*8,8,pkey,key);

        /* Check/Generate ODD Parity */
        for (i=0;i<(len*64);) {
                parity = 1;
                for (j=0;j<7;j++,i++)
                        parity = (parity + key[i]) % 2;
                if ((sw == 1) && (key[i++] != parity))
                        return(FALSE);
                if (sw == 2) key[i++] = parity;
        }

        if (sw == 1) return(TRUE);

        /* Pack Key */
        pack(len*8,8,sptr,key);
        return(TRUE);
}
```

```
/*****************************************************************
 *  PACK()  Procedure to PACK a binary block consisting of       *
 *  1 bit per byte into an equivalent block consisting of        *
 *  'len' bytes packed 'bpw' bits per byte.                      *
 *****************************************************************/

pack(len,bpw,packed,binary)
int len;        /* length of packed block in bytes */
int bpw;        /* number of bits packed per byte */
char *packed;   /* the packed block (ie. bpw bits/byte) */
char *binary;   /* the unpacked block (ie. 1 bit/byte) */
{
        register i,j,k;

        for (i=0;i<len;i++) {
                k = 0;
                for (j=0;j<bpw;j++)
                        k = (k<<1) + *binary++;
                *packed++ = k;
        }
}


/*****************************************************************
 *  UNPACK()  Procedure to UNPACK a 64-bit binary block stored   *
 *  in 'len' bytes with 'bpw' bits per byte into equivalent      *
 *  1 bit per byte form.                                         *
 *****************************************************************/
unpack(len,bpw,packed,binary)
int len;                /* length of packed block in bytes */
int bpw;                /* number of bits packed per byte */
char *packed;           /* the packed block (ie. bpw bits/byte) */
char *binary;           /* the unpacked block (ie. 1 bit/byte) */
{
        register i,j,k;

        for (i=0;i<len;i++) {
                k = *packed++;
                for (j=0;j<bpw;j++)
                        *binary++ = (k>>(bpw-j-1)) & 01;
        }
}

/*****************************************************************/
```

OTHER METHODS OF DATA ENCRYPTION

Although software can be used to perform the electronic codebook mode encryption and decryption functions of the DES, recall that this mode is defined in the standard only for electronic hardware. This is important in two cases. First, if there is a requirement that the protection application be in full accord with the standard, electronic devices must be used to implement the encryption. Second, if the application requires the repeated protection of entire files for transmission, the use of electronic encryption devices will likely be more efficient than the use of software encryption.

PLUG-IN CIRCUIT BOARDS FOR PERSONAL COMPUTERS

Data encryption devices in the form of DES plug-in circuit cards for personal computers are available from a number of manufacturers. Depending on features offered, these boards cost between $250 and $1,600 each. Some of the vendors that offer boards incorporating the DES are Okiok Data (Laval, Quebec), Winterhalter Inc. (Ann Arbor, Michigan), Jones Futurex Inc. (Rancho Cordova, California), MPPI Ltd. (Glenview, Illinois), and Vutek (San Diego, California).[1]

These devices provide varying levels of access and data protection. Typically, the user is able to specify a different key for each file to be encrypted. If desirable, a group of files (or all files on a drive) can be encrypted with the same keyword. Files can be encrypted and transmitted between computers in their encrypted state. Decryption would be done at the receiving site and the original file recaptured for use. If data are to be transmitted in encrypted form, it would be wise either to have the same brand of device at both ends of the line or to have tested them for compatibility. Different treatments of keyword parity, mode of DES operation, and padding short final blocks, all within the official scope and framework of the DES framework, will result in different encryptions. In such a case, not even the proper keyword could recover the original data from the enciphered files.

MAINFRAME DATA PROTECTION

Computer centers will often have available to users one or more procedures to encipher and decipher sensitive data. The

---

[1]GAO has not examined or in any way evaluated any of these products. Their listing here does not constitute an endorsement by GAO either of the product or the company. Inasmuch as many other companies offer similar products, this listing is illustrative only. This caveat extends to all other products mentioned in this appendix.

method used to scramble the data may, or may not, be a software implementation of the DES. Typically, however, it will require a keyword to be supplied by the user and will then make use of the keyword to encrypt the records.

For example, the NIH computer center provides a set of subroutines to encipher data files. The user specifies the file to be encrypted and supplies a phrase to serve as a keyword. The keyword is used to generate a sequence of random numbers that are used to encipher the file character by character. Other centers provide mainframe subroutines to encipher files, records, or parts of records. In all cases, the keyword is maintained outside of the mainframe for security. The cautions in user manuals are usually quite blunt. Data should only be protected by encryption if it can be recreated. If the key is lost, the data is also lost.

There are also large data security software packages that would be used by a company if everything were to be protected, albeit to differing levels. An example of one such package is "Top Secret," offered by Computer Associates International for IBM mainframe computers operating under the multiple virtual storage environment.

OTHER SOFTWARE PROTECTION

Many types of software protection are available. Packages such as SuperKey, produced by Broland International Inc., incorporate encryption algorithms to encipher microcomputer files.[2]

Computer journals, both professional and popular, often publish articles on data encryption. These articles range from technical discussions on the strength of encipherments to programs in the BASIC language that will encipher words or files. Some methods are based on very hard mathematical problems that have no known general solution, for example, decomposing large numbers into prime factors. Other methods provide a moderate amount of protection against intrusion.[3]

---

[2]SuperKey has two encryption algorithms available. One uses a proprietary algorithm for fast encryption and incorporates the file name as part of the encryption process. The other procedure makes use of the DES encryption algorithm.

[3]For discussions of other type of cryptographic systems, see, for example: R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, 21:2 (February 1978), 120-26 and W. Diffie and M. Hellman, "Privacy and Authentication: An Introduction to Cryptography," Proceedings of the IEEE, 67:3 (March 1979), 397-427.

RECENT DEVELOPMENTS IN DATA ENCRYPTION

The National Bureau of Standards (NBS) has supported the DES in a published federal information processing standard entitled Data Encryption Sandard (FIPS 46) and has encouraged its use by civil agencies and the public. NBS issued this standard in 1977, as part of its role in establishing unclassified information standards pursuant to the Brooks Act (public law 89-306) and Executive Order 11717. In 1983, the DES was reapproved by NBS for federal encryption purposes until 1987. NBS is one of four federal agencies with responsibilities for data encryption policy and procedures. The others are the Office of Management and Budget (OMB), the General Services Administration (GSA), and the National Security Agency (NSA), which has been given increasing responsibility in recent years. The mandates of these four agencies have come under increasing scrutiny in the last several years. Important questions have arisen concerning the potential overlap of their roles, and more specifically, the advisability of using various hardware and software solutions for different types of data encryption needs.

In 1984, the President signed national security decision directive (NSDD) 145, which gives the responsibility for protection of classified, national security information, and "other sensitive, but classified . . . information . . . the loss of which could adversely affect the national security interest . . ." to the National Telecommunications and Information Systems Security Committee (NTISSC). Under the same directive, the director of NSA is assigned the role of national manager for telecommunications and automated information systems security.

NSA has indirectly raised objections to the DES that involve its nonsecret nature. Since NSA does not certify any software application, it has not considered the software application of DES in this transfer paper. NSA requires a very high level of secrecy in all its encryption procedures. The algorithm on which the DES is based is not secret, but is published and available. NBS supports the DES, in part, because it is apparently not decipherable except under extraordinary circumstances. The expertise necessary to decrypt a DES-encoded data set would be at the highest level, and the expense involved, equally high. It can be argued that only defense agencies would be able to shoulder the burden of this cost. More generally, testimony in 1985 by Milton J. Socolar, Special Assistant to the Comptroller General, points to the potential problem of cost in using DOD-defined standards for computer secrecy:

> There is a potential that we will commit ourselves to the development and teaching of inordinately expensive approaches to computer security. DOD, in its approach to security, seeks to counter identified or perceived threats to the national defense, treating costs as a

decidedly secondary factor in determining the degree of protection required. The National Bureau of Standards, on the other hand, emphasizes a risk-management approach that uses cost as a primary determinant.

To ensure the maximum attainable level of security, NSA supports hardware applications that it certifies. NSA will work with manufacturers to produce hardware devices for encryption of classified, national security information and nonclassified information that may affect the national security. NSA will assist manufacturers in developing encryption hardware including tamperproof "black boxes" that are used to provide for the security of the information being processed. NSA will probably retain all possible keys to such hardware devices if they are related to the protection of national security matters.

Considerable thought has been given to the requirements of protecting classified, sensitive, military data vs. the requirements of protecting information not related to the national security interest. For some practical purposes, the DES provides an adequate level of protection for information used in many evaluation and audit applications, except the security-related applications as discussed on pages 10-11 of this transfer paper. While the DES is based on a published algorithm, it is not thought to endanger the safety of certain types of data which are encrypted using this standard because of the extensive, expensive efforts that would be required to decipher such information. It should also be added that software encryption using the DES results in encryption that is identical to hardware encryption based on the DES, if the software is appropriately re-checked and secured with each application. With "reasonable resources" available outside defense agencies, DES is virtually unbreakable.

Finally, it should also be mentioned that there has been considerable debate in the last several years concerning the entry of NSA (through the influential role of the director of NSA as the national manager of systems security under NSDD 145) and other defense agencies into areas other than classified, national security matters. It is not a simple matter to define the boundaries between "classified national security information," "other sensitive, but unclassified information, the loss of which could adversely affect the national security interest," and still other sensitive information. However, for some audit and evaluation procedures the software encryption tool provided in this transfer paper is a potential encryption mechanism. GAO evaluators could use the tool for the encryption of some information at civilian agencies, which does not require certified hardware solutions. When there is ever doubt whether information is related to the national security interest, an appropriate approach might be to encrypt the data according to NSA guidelines. GAO evaluators should be mindful of the current debate between NSA, NBS, OMB, and GSA, and aware that overlapping

responsibilities raise serious questions in the minds of many federal administrators and legislators. The software tool based on the DES that is provided in this document could be considered adequate for some purposes unrelated to the national security interest that do not require certified, hardware solutions.

Individuals desiring further background on the current policy debate are directed to a February 1986 publication by the Office of Technology Assessment: Federal Government Information Technology: Management, Security, and Congressional Oversight, and especially to the section entitled "Major Findings" in that document.

BIBLIOGRAPHY

Athanasiou, T. "DES Revisited." Datamation, 31:20 (October 15, 1985), 110-14.

Boruch, R. F. "Strategies for Eliciting and Merging Confidential Social Research Data." Policy Sciences, 3 (1972), 275-97.

Chaum, D. "Security Without Identification: Transaction Systems to Make Big Brother Obsolete." Communications of the ACM, 28:10 (October 1985), 1030-44.

Diffie, W. and M. Hellman. "Privacy and Authentication: An Introduction to Cryptography." Proceedings of the IEEE, 67:3 (March 1979), 397-427.

Gait, J. Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard. U.S. Department of Commerce, National Bureau of Standards, NBS special publication 500-20. Washington, D.C.: September 1980.

GAO (U.S. General Accounting Office). High-Quality Senior Marine Corps Officers: How Many Stay Beyond 20 Years of Service?, GAO/PEMD-85-1. Washington, D.C.: November 1984.

-----. The Special Supplemental Food Program For Women, Infants, and Children (WIC): How Can It Work Better?, CED-79-55. Washington, D.C.: February 1979.

Kolata, G. "Flaws Found in Popular Code." Science, 219:4583 (January 28, 1983), 369-70.

Office of Technology Assessment. Federal Government Information Technology: Management, Security, and Congressional Oversight. Washington, D.C.: February 1986.

Rivest, R., A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." Communications of the ACM, 21:2 (February 1978), 120-26.

U.S. Department of Commerce, National Bureau of Standards. Data Encryption Standard, federal information processing standard (FIPS) publication 46. Washington, D.C.: January 1977.

-----. DES Modes of Operation, FIPS 81. Washington, D.C.: December 1980.

-----. Guidelines For Implementing and Using the NBS Data Encryption Standard, FIPS 74. Washington, D.C.: April 1981.

-----. Computer Data Authentication, FIPS 113. Washington, D.C.: May 1985.

GLOSSARY

Algorithm. A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.

Alphanumeric. A character which may be alphabetic, numeric, or a special symbol.

ASCII. American standard code for information interchange. A standard 7-bit binary representation of the characters used for information interchange among data processing systems.

Bit. A single digit in the binary number system, i.e., the digit "0" or "1."

Block. A group of eight consecutive bytes (i.e., a string of 64 binary digits). The basic input structure to the DES algorithm in the electronic codebook mode.

Byte. A sequence of eight adjacent binary digits operated on as a unit. Often used to refer to the computer representation of a character.

Ciphertext. The transformed message in secret form. See plaintext.

Cryptography. The process of rendering a message unintelligible to outsiders by various tranformations of the symbols used to express or transmit the message, e.g., the substitution of letters, numbers, or characters for other letters, numbers, or characters.

EBCDIC. Extended binary code for data interchange. A binary representation of characters that is used primarily by IBM.

Encryption. The process of changing plaintext into ciphertext. "Encoding" is a synonym. Also see cryptography.

Hexadecimal. A number system in which the symbol "10" has the decimal value "16." The sequence of single-digit hexadecimal counting symbols is: 1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Frequently used in computer applications because one byte is two hexadecimal digits.

Key. A sequence of symbols (bits, hexadecimal digits, or alphanumeric characters) that determines the precise series of transformations used in the encryption process. A key is normally selected by the user of the encryption process.

Look-up table. A collection of data in a form suitable for ready reference, frequently stored in the form of an array of rows and columns.

Plaintext. A uncoded message (e.g., unencrypted data) that is readable by an observer. Normally refers to the message to be put into (or recovered from) secret form.