Cryptographic methods of data protection have taken on new importance as computers have become faster and as strong cryptographic algorithms, such as the Data Encryption Standard (DES), have become available. But a standard encipherment technique is only the first step in applying cryptography in a computing center. This paper discusses the Information Protection System (IPS), *a set of cryptographic application programs designed to use the DES algorithm in a working computing center. In designing IPS, several important augmentations of DES were formulated. IPS was first implemented to help increase computing-center security at the IBM Thomas J. Watson Research Center and is now widely installed at other IBM locations. IPS is not an IBM product and is not available for use outside IBM, but many cryptographic techniques in IPS were incorporated into the IBM cryptographic products announced in 1977.*

The IPS cryptographic programs

by A. G. Konheim, M. H. Mack, R. K. McNeill, B. Tuckerman, and G. Waldbaum

In the third quarter of 1974, IBM's Research Division began an investigation of ways to improve computing-center security at the Thomas J. Watson Research Center in Yorktown Heights, New York. Several steps were taken to improve physical security. Transparent protective walls were constructed to make access difficult for unauthorized persons. Procedures for changing combinations on pushbutton-code locks were tightened, and other measures of a similar nature were adopted.

Nevertheless a serious problem remained: Behind the locked doors and protective walls were computing systems with security exposures not related to their physical surroundings. For example, except for the most highly classified data, any tape could be mounted by knowledgeable users of our computing systems, which shared a common operations staff and tape library. In one system, OS/MVT,[1] any on-line data set could be read by any job. And in VM/370,[2] as later shown by Attanasio, Markstein, and Phillips,[3] the system could be penetrated by a determined attacker.

Several alternatives were considered for improving the security of on-line and off-line data at Yorktown. The OS/MVT *password* facility was considered for data sets but was rejected because of its operational inconvenience and because the demonstrated weakness in VM/370 made it difficult to justify serious inconvenience in OS/MVT. If a "protected" OS/MVT volume could easily be mounted by a VM/370 user, no real protection existed. A manual record-keeping system to prevent unauthorized mounts in VM/370 was also rejected as awkward and error-prone. Further, it was concluded that confidential data could not be considered really safe if it could be accessed by operators, system programmers, and other users with privileged access to the system.

To meet the needs of users who held confidential data in this environment, a cryptographic system called the *Information Protection System (IPS)* was developed. By using IPS programs to apply a cryptographic transformation to their data, users have been able to protect confidential information against unauthorized release. Data in its original, ordinary form—such as the text of this paper—is termed *plaintext* or *cleartext*. After a cryptographic transformation has been applied—that is, after the data has been *enciphered*—the resulting data is termed *ciphertext*. The reverse of encipherment—returning ciphertext to its original plaintext form—is *decipherment*. While encipherment does not protect a file against accidental or malicious destruction, the owner can be confident that the information in an IPS-enciphered file will almost certainly never be read in plaintext by an opponent, nor will an attempt to modify the data in secret be successful.

The IPS cryptographic system consists of two types of information: public and private. By *public information* we mean the type of system employed and the details of its operation. The design of IPS cannot be kept secret because any user of the computing system can copy the IPS programs and sooner or later arrive at an understanding of the method used. (A cryptographic system implemented in hardware is not so readily probed, but in our view the result is the same: the techniques used can become public knowledge.) A cryptographic system that depends for its secrecy on an opponent's ignorance of its method of operation will therefore not be successful. To provide the essential element of secrecy, users are required to supply privately known strings of characters or bits, termed *keys*. A key is used to select a transformation from a family of cryptographic transformations, one for each possible key. With a well designed cryptographic system, knowledge of both the system *and the key* is required to obtain plaintext from ciphertext.[4]

The IPS cryptographic programs use a *keyed block cipher*. A user is required to provide a key in order to obtain encipherment or decipherment services. The keys are not stored in the computing system, an arrangement that has both advantages and disadvantages. The user is inconvenienced by having to supply a key for each use of IPS, but he has the assurance that no one, not even a privileged machine operator or system programmer, can decipher his files unless his key can be intercepted while in active use. Because keys are not stored in the system, IPS-enciphered data resists attack even if opponents masquerade as legitimate users and employ the IPS system itself to aid them in their work.

IPS-enciphered data is now used in our computing systems in substantial quantity, and its owners still enjoy convenient access to their files. Although IPS does not provide an absolute answer to the problems of computing-center security, it has been in use at Yorktown and has been delivered to many IBM locations over the course of more than four years.

The basic cryptographic transformation used by IPS is that specified by the Data Encryption Standard (DES),[5] which was in the process of being approved by the United States Government when IPS was designed. Much cryptographic work on the algorithm had been done at the Research Center in Yorktown Heights[6-8] and at IBM's System Communications Division laboratory in Kingston, New York.[9-11] In a DES encipherment, a 64-bit (eight-byte) block of plaintext data is transformed, under the influence of a 56-bit cryptographic key, to a 64-bit block of ciphertext data. (The full key is 64 bits, but only 56 participate in the encipherment. The others are parity bits.)

By enciphering data eight bytes at a time instead of one at a time, DES greatly increases the number of possible cryptographic substitutions; but either all data must fit the algorithm's eight-byte length, or some compensating arrangement must be made. The IPS method of encipherment augments DES to permit the encipherment of data of any length by using a method of successive data-dependent encipherment called *chaining*.

In our view, the real usefulness of IPS, going beyond the basic cryptographic function provided, is the protection offered in the user's natural programming environment.

Design philosophy of IPS

The goal of IPS is to offer an implementation of the Data Encryption Standard that is easy to use while providing the Standard's full cryptographic strength, as well as several additional features. The design principles of IPS are as follows:

key management

No key, or variable that is equivalent to a key, resides permanently in the computing system. Keys are the responsibility of *each user; they are entered into the system at the time of encipherment or decipherment. Keys are thus exposed in the system only during residence of the job or command that uses* IPS*, not on a permanent basis.* A cryptographic system that links two users must provide a mechanism for establishing a common operational key. If one of the users is a host operating system, some provision must be made either for storing keys in the host or for the host to construct the operational key. If keys are permanently resident in the system, then users with privileged status may be able to obtain them. While such users might possibly place traps in an operating system to recover keys during an IPS session, this exposure is smaller, and the attack is probably more dangerous to the attacker than inspection of a permanently resident file of keys.

key formats

A variety of key formats is offered to users. The DES algorithm specifies a 64-bit key but makes use of only 56 bits, with eight bits serving as parity check bits. IPS allows the direct entry of eight-byte DES keys expressed in hexadecimal notation. For maximum safety, the 56 active key bits should be chosen randomly, but randomness is not always easy to achieve, even by experienced users. Therefore, in addition to direct DES key input, IPS allows a user to specify the key as a long character string to be converted internally to a form suitable for DES. We believe that a key expressed as a character string is easier for a user to invent and remember, and if chosen with reasonable care it can be the equivalent of a random DES key. (Users should avoid familiar names, numbers, and phrases, and they should use long keys when the individual key characters are not selected randomly. These matters are discussed under *Key selection and key crunching*, below.)

record size preservation

Although DES *requires eight-byte inputs,* IPS *is so designed that data records whose length is either less than eight bytes, or not a multiple of eight bytes, are accepted and do not increase in size under encipherment.* Users are not required to pad their data to conform with DES, nor do the IPS programs lengthen data in this way. Altered data lengths would unnecessarily complicate the computing environment by requiring changes in data set attributes and in application-program array sizes and string lengths.

repetitive plaintext patterns and chaining

Repetitive patterns in plaintext are not mirrored in the ciphertext. DES is a block cipher which enciphers identical eight-byte plaintext blocks into identical eight-byte ciphertext blocks (under the same key—an assumption made throughout). By employing *chaining* in IPS, it is possible to use DES in such a manner that identical eight-byte plaintext blocks virtually always yield different ciphertext blocks, without altering the security of the existing algorithm. (Our interpretation of *virtually always* is discussed below, under *Block chaining*.)

IPS facilities

Utilities and commands are provided for handling OS/VS data sets and VM/370 CMS files,[12,13] *and data can be enciphered and deciphered from within user programs in FORTRAN, PL/I, and Assembler. Ciphertext files produced by the utilities and by user programs are compatible, provided that users follow IPS conventions when creating files to be deciphered by the utilities, or deciphering files created by the utilities. For nonsequential access methods not supported by the utilities (for example VSAM and BDAM*[14,15]*), encipherment from within user programs still permits the creating of ciphertext files. When needed, the use of IPS within user programs permits the designing of applications that read and write data sets entirely in ciphertext, without ever having to expose files of plaintext.* It is important to provide a service that is sufficiently flexible to keep users satisfied—that is, pleased at the way IPS fits into their environment and content with the investment of time and effort required to introduce cryptography into their applications. Therefore IPS supports both batch and interactive use, and it makes both utilities and subroutines available. In designing IPS, it was not feasible that the utilities be made to handle all conceivable types of data sets (at least with the development time available to us), so the subroutines provide an important mechanism for enciphering files of all types because the user programs themselves will handle I/O. And if user programs read and write in ciphertext, plaintext files need never exist.

header record

IPS-enciphered files ordinarily contain a header record to identify and describe them as IPS ciphertext. The record is inserted and removed automatically by the utilities, and it is passed back to those who use IPS from within their own code. The header record includes *(i)* information about the type of encipherment used, *(ii)* a time-date stamp, *(iii)* the version of IPS employed, *(iv)* cryptographic chaining information, *(v)* a *verification* field to warn a user at time of decipherment when an incorrect key has been supplied, without providing information that enables the correct key to be recovered, and *(vi)* an optional *user comment* field.

file exchange

To the extent that plaintext files can be exchanged between OS/VS and VM/370 CMS systems, the corresponding IPS-enciphered files are also exchangeable, with encipherment in one system and decipherment in the other, as needed.

programming and documentation

The programs were designed and coded with great care, since incorrect output is not easily detected in ciphertext. Similarly, because misuse of the programs can cause loss of user data, and worried users may not submit their data to cryptographic transformation, the documentation has been made as clear and helpful as possible.

Augmentation of DES by chaining (and key-crunching)

Certain plaintext files exhibit great regularity, with many identical eight-byte blocks (for example, blocks of eight blanks to fill out lines of computer source programs to a fixed length). Normally under DES encipherment, such identical blocks of plaintext yield identical blocks of ciphertext (under the same key). Thus the eight-byte blocks of blanks may be identifiable, and a rough geometric outline of regions of nonblank characters among blank characters in the plaintext may be discernible in the ciphertext. Repetitions of some other blocks may also be visible: for example, identical records in the plaintext, or identical parts of records when aligned with the eight-byte DES blocks, can be recognized as identical in the ciphertext. It is doubtful whether this phenomenon is a serious weakness. To date no technique has been found that uses it to determine the key or to obtain usable plaintext of alphabetic or numeric files (although some of the structure of digitized line drawings could be visible). However, if the existence of identical blocks can be concealed, a cryptographic system is strengthened at least intuitively because the amount of information available to an opponent is reduced. Because of the possibility that under some circumstances, with some ciphertexts, an opponent might be able to make use of repetitions, an augmentation of the DES algorithm was deemed advisable in IPS.

After all the eight-byte blocks of a record have been enciphered using DES, there frequently remains a *short block*, a block of fewer than eight bytes. A short block cannot be enciphered directly by DES, since DES requires eight-byte inputs. It could be padded on the right with zeros or blanks and then enciphered, but all eight bytes of the resulting ciphertext, and preferably information as to the length of the padding, would have to be preserved for future decipherment, and the ciphertext would be longer than the original plaintext. We found that condition undesirable because, for example, the ciphertext might have to replace the plaintext in some previously allocated space, or be written according to some previously defined record length. Alternatively, some key-dependent "simple substitution" encipherment could be designed for short blocks; but if it depended only on that block (and the key), it could be subject to cryptanalysis, especially when the length of the block is preserved. This exposure, although not so obvious as the repetition of identical blocks, is potentially more serious because some nonblank short-block plaintext could be discovered.

Chaining

To handle both types of problems, we devised a method called *chaining*. Chaining is a process by which each block of ciphertext is made to depend not only on the corresponding plaintext and the key, but also on preceding ciphertext. If the dependence starts anew with each record, the process is termed *block chaining*; if it continues across record boundaries, it is termed *record chaining*. Augmentation of DES by chaining eliminates repetitiveness in ciphertext arising from repetitive plaintext; it provides for the encipherment and decipherment of data of arbitrary length (not necessarily multiples of eight bytes); and it allows the user to enter keys as long character strings.

In the discussions of block and record chaining that follow, there is a possibility of confusion in our use of the terms *block, data string*, and *record*. When we speak of enciphering a *record*, we tacitly assume that the data to be enciphered is a file on some storage medium and is organized into records, and that the natural logical unit of encipherment is a record. The IPS subroutines can also be used on data that is not so organized, for example in main processor storage; and the logical unit of encipherment may be some *data string* which is defined by the programmer. It might for example be a selected portion of each record, the rest being left unenciphered. Thus, in what follows, when we use either *record* or *data string*, in most cases the other term could be used as well. In addition, as used here, a *block* is an eight-byte unit of data suitable for DES encipherment or decipherment. It should not be confused with the physical data blocks of a file on a tape or disk volume, nor with the reproduction of those blocks in input or output buffers.

block chaining

We term the basic chaining operation *block chaining* because it chains together successive eight-byte blocks of DES data. A formal description of block chaining is given in the Appendix. Below is a description of the operation as implemented in IPS, first assuming that the length of each record (or data string) is some multiple of eight bytes.

First, once for the entire file, a time-dependent eight-byte *initial chaining value (ICV)* is generated. Then, for each record:

- The initial chaining value is assigned to an eight-byte *current chaining value*.
- The current chaining value (equal to the ICV) is XORed[16] with the first plaintext block. The result is enciphered with DES, and the consequent ciphertext is assigned to the first block of output.
- The just-produced ciphertext block is also assigned to the current chaining value.
- The current chaining value (equal to *ciphertext*) is XORed with the next plaintext block. The result is enciphered with DES, and the consequent ciphertext is assigned to the next block of output.

- The just-produced ciphertext block is also assigned to the current chaining value.
- The last two steps are repeated for each additional block of plaintext.

Chaining can be viewed as a wave of data modification and encipherment that moves from left to right along the string of data being processed. Because each eight-byte block of plaintext is modified before it is enciphered, and each modification is different, the resulting ciphertext blocks in a record are (almost certainly) different whether the plaintext blocks are identical or not.[17] A data string composed of ten identical eight-byte blocks is enciphered into ten *different* eight-byte blocks of ciphertext.

Decipherment with chaining is the inverse of encipherment with chaining, as follows: Each block of ciphertext is deciphered with DES, and the result is XORed with a current chaining value to yield a plaintext block. For the first ciphertext block in a record, the current chaining value is the same as the initial chaining value used during encipherment. For each subsequent ciphertext block, the current chaining value is the immediately preceding ciphertext block.

Because ciphertext (rather than plaintext) is used for the current chaining value during encipherment, this method of chaining has an important "self-healing" property. Chaining as described above might appear to threaten the destruction of all data in a string to the right of some point of disturbance, as from an I/O error. But nothing of the kind happens. Although each ciphertext block depends *implicitly* on all the preceding blocks in the record, as well as on the current plaintext block, it depends *explicitly* on only the immediately preceding ciphertext block and the current plaintext block (and the key), as follows:

ith ciphertext block =
DES{key, $(i - 1)$st ciphertext block XOR ith plaintext block}

From this equation we see that

ith plaintext block =
$(i - 1)$st ciphertext block XOR DES^{-1} {key, ith ciphertext block}

Thus a plaintext block can be recovered from just the current and immediately preceding ciphertext blocks (and the key). Under this type of chaining, therefore, any damage is not propagated beyond the two blocks at the point of difficulty.

short blocks Since short blocks cannot be enciphered by DES without increasing their length, another method of equal strength is required for enciphering them. The method we chose is to XOR the short block with a *variable, secret* quantity. Such a quantity is made available by slightly modifying the chaining process.

To encipher a *trailing* short block of $l < 8$ bytes—that is, a short block that follows one or more full blocks in the record—the preceding full block of ciphertext is *re-enciphered*, and the first l bytes of the result are then XORed with the plaintext short block. That preceding full block of ciphertext depends on all the preceding blocks of the record, and thus is sufficiently variable. But it is visible to an opponent; re-encipherment of it provides the necessary secrecy. By this means, trailing short-block ciphertext has the full strength of a standard DES encipherment.

However, to encipher (under block chaining) a *record* of $l < 8$ bytes—that is, a *short record*—it is XORed with the first l bytes of the encipherment of the initial chaining value, since there is no "preceding ciphertext block." This encipherment is *not* strong,[18] but it does superficially conceal the short records. There is no impairment of the strength of longer records. Under record chaining (see below), which we recommend, and which *is the default action*, this problem disappears. It has been suggested that the programs should prohibit this block-chaining-but-not-record-chaining encipherment of short records. However, virtually the only short records in our experience have been formatting command symbols in literal text (for example, *.sp* for "line-space" in SCRIPT files), and those symbols carry little information that would be useful to an opponent. Therefore we felt that such a prohibition might be unnecessarily burdensome to some users, as well as possibly awkward logistically. For files that contain *significant* short records—for example, if all records are short—the user should of course avoid overriding the default.

record chaining

Block chaining still has two potential limitations. First, identical *records* in a file have identical encipherments because the use of the same key, data, and initial chaining value yields the same ciphertext. In fact, two records with a given number of identical initial blocks yield ciphertexts that agree in the same number of initial blocks. Second, if enough short records exist in the file, it may be possible, as noted above, to decipher them (but not any full blocks) without the key. Both limitations disappear if the initial chaining value is allowed to vary from record to record throughout the file. Thus if each record is enciphered with its own unique initial chaining value, all the ciphertext blocks are different, even if *all* the plaintext blocks of *all* the records are the same. And the cryptographic weakness mentioned above for short records is removed.

This variable initial chaining value could be chosen by a function that does not depend on the contents of the data records (as by adding a constant to the preceding initial chaining value), but if any data records were lost, the synchronization between the records and the initial chaining values would also be lost. We chose a method, outlined below, that gives good variability, and that depends only on the ciphertext of one prior record, or rarely of a few prior records. Assuming a fixed key, encipherment of a record under the record chaining option requires two arguments: the plaintext record and the eight-byte initial chaining value. The encipherment yields two results: the output ciphertext record and an *output chaining value* associated with the record. Following is a description of record chaining as extended from the underlying block chaining:

- A *starting* initial chaining value is generated the same way as in block chaining.
- The first record is enciphered exactly as in block chaining.
- An output chaining value (OCV) is formed by taking the rightmost eight bytes of the concatenation (||) of the initial chaining value (ICV) and the just-constructed ciphertext record:

  OCV = RIGHT8[ICV||ciphertext]

- The output chaining value is used as the *input* chaining value (the ICV) for enciphering the next record.
- Chaining continues in this manner for the remaining records in the file.

Thus with record chaining, the chaining process continues across record boundaries, effectively hiding duplicate plaintext, strongly enciphering even short records, and doing these with a self-healing property similar to that for block chaining.[19] To correctly decipher a record (given the key), only that record and the preceding eight bytes of ciphertext (or all of them and the starting initial chaining value if there are fewer than eight bytes) are required. In the usual case, in which the preceding record is not short, the preceding eight bytes are the last eight bytes of the preceding ciphertext record; in the most extreme case, eight preceding records of one byte each may be needed.

Normally, record chaining is used for sequentially organized files because as each record is processed in turn, it defines the initial chaining value for the next. Block chaining is best used for nonsequential files, in which there is no fixed order of accessing records. IPS strongly encourages the use of record chaining by making it the default option, but users can easily select block chaining instead. Even for files that are to be updated by random access, the beneficial variability of record chaining can be obtained, without loss of information due to random-access updating, by beginning each record with an eight-byte *throwaway block* which need not be correctly deciphered.

Key selection and key crunching

In the past there has been a tendency among users of cryptographic systems to use a *word* or a *name* as a key. Today this procedure is unsafe. There are fewer than a million words in an unabridged dictionary, and fewer than a million names in a large telephone directory. That number of potential keys could be tested in a few minutes on a System/370 Model 168. Even the use of eight randomly chosen EBCDIC characters is unsafe, because only about a third of the 256 EBCDIC characters can be printed at a terminal, which reduces the available keys by a factor of about 1.5×10^{-4}. Instead, if a DES key is introduced *directly*, it should consist of *16 hexadecimal digits, chosen randomly or essentially so* (except for parity.)

IPS provides for an additional way of defining a key. It permits the user to enter a *user key* of more than eight bytes, say 16 bytes or more (usually, but not necessarily, printable characters and blanks). With this freedom, the user might, for example, enter several words (say five or more) chosen *randomly* and *independently* from an unabridged dictionary. Or the user might enter some phrase. It is essential that these choices be made in such a way that the key cannot practically be found by an opponent, either by guesswork or by enumerating some plausible set of keys.

To produce a suitable DES key from such a longer user key, some good *hashing* function is needed. It should be sufficiently complicated to produce essentially unbiased and statistically independent bits in the DES key. It would not be desirable, for example, merely to XOR various bytes together to form the DES key. The EBCDIC representations of decimal digits and capital letters all commence with binary 11, hence the XOR of any number of these commences with 11 or 00, so that the leading two bits of such an XOR are perfectly correlated, thereby limiting the set of possible keys. More generally, any linear combination of biased or correlated bits will have a non-uniform distribution, which might be useful to an opponent by allowing him to search only over the more likely keys.

We solve this problem by the use of chaining. IPS enciphers the long user key under a selected key, with chaining, and it uses the rightmost 56 bits of the resulting ciphertext as the DES key. We call this procedure *key crunching*. We believe that correlations in the user key are adequately smoothed out by crunching. The DES key can be returned to the user so that either it or the long key can be used for future decipherments.

Components of IPS

The OS/VS version of IPS has two components: a utility program called IEBCODE for enciphering (or deciphering) sequential data sets, and a set of cryptographic subroutines intended for calling from Assembler, FORTRAN, and PL/I Optimizing Compiler programs. In addition, an interactive CIPHER command is provided for TSO (the Time Sharing Option) running under MVS (Multiple Virtual Storage).[20,21] The VM/370 CMS version of IPS has two components: an interactive CIPHER command used to process CMS files and the same set of subroutines used in OS/VS.

The IEBCODE utility program

IEBCODE is a cryptographic utility for sequential data sets. That is, it produces an enciphered copy of a sequential plaintext file, or a deciphered copy of a sequential ciphertext file. Individual members of a partitioned data set can also be processed, provided that it is not necessary to preserve the user data fields, if any, in the partitioned-data-set directory entry. If other types of data sets are to be protected, then either the data must be unloaded (by one of the normal utilities) and the resulting sequential data set enciphered by IEBCODE, or, better, the data sets can be processed by user programs that read and write the data sets themselves and call IPS subroutines for cryptographic services.

logical record encipherment

IEBCODE can process any data set accessible by the Queued Sequential Access Method (QSAM).[15] All valid fixed-length, variable-length, variable-spanned, and undefined-length data sets can be processed. IEBCODE does not allow a change of record format, but it permits the reblocking of data as requested by the user in the Job Control Language DD statement DCB parameter.

IEBCODE enciphers logical records individually, rather than in any larger or different unit, for compatibility with user application programs that make use of IPS subroutines. Because both the subroutines and IEBCODE were to be used as part of the same package, we felt it important that the exchange of data between them be relatively easy. Further, for applications that must be as secure as possible, we intended that the IPS subroutines be used in preference to IEBCODE, or used after IEBCODE had converted all the old plaintext secret files to ciphertext. We believed that many of the most important applications of IPS would use subroutines almost exclusively because under the subroutines, plaintext external files need never exist: instead, the data is left in ciphertext until the moment that each data item is to be used in main storage by a running program. It is important that the two parts of IPS be compatible in order to support this type of application, in which IEBCODE is used for the initial bulk conversion of data files from plaintext to ciphertext, or used occasionally to create ciphertext files from newly arrived data, while the subroutines would be used for updating. IEBCODE, therefore, had to be designed to produce ciphertext that could easily be read and processed by user programs.

Data to be deciphered by a subroutine call is handled most easily if ciphertext records are processed one at a time, in the same manner as the processing of ordinary records by an application that does not use cryptography. This natural processing of data set records by an application program is possible only if the unit of encipherment, the data enciphered in one logical operation, is exactly one logical record. It cannot, for example, be a group of records, because such a technique would force the application program to read and process records in groups. For most applications, substantial program changes would be required to adopt such a scheme. Therefore, under IEBCODE, each record in the data set is enciphered separately, so that ciphertext produced by IEBCODE can easily be read and processed by user programs.

For fixed-length records, the unit of encipherment is the logical record. For variable-length records, the unit of encipherment is the data portion of the logical record; the record descriptor word remains in plaintext, to be accessible to data management routines. For undefined-length records, the unit of encipherment is the physical data block, because user programs must also treat each such block as an individual record. For spanned variable-length records, the unit of encipherment is the data portion of the entire spanned record, no matter how many segments or physical blocks it may occupy. By making the unit of encipherment for spanned records the entire record rather than a record segment, IEBCODE ensures that the ciphertext produced is independent of block size and record segmentation.

For users of IEBCODE who employ it only for the bulk encipherment and decipherment of data, logical record encipherment is of no particular importance. But that technique makes it possible for users to exchange data between IEBCODE and their own cryptographic application programs with minimal difficulty. This design also permits the exchange of files between OS/VS systems and the IPS CIPHER command running under VM/370 CMS.

Logical record encipherment can be set aside if the benefits of incompatibility exceed the costs. For instance, it is possible to imagine a scheme in which incoming plaintext records are processed by a data compaction routine to save space and the CPU time needed to encipher them, then are placed in successive portions of a standardized fixed buffer, enciphered, and transmitted to an output data set. Such a compressed and enciphered file theoretically could be processed by a user program, but that is not likely to happen for practical reasons. For most purposes, and in the absence of special reasons as in the above example, the logical record encipherment technique is preferred.

encode ID record

Enciphered files are identified as such by IEBCODE. A header record, the *encode ID record*, is written at the front of each file of ciphertext. This ID *(identification)* record contains the date and time of the run, the version of IPS used, a security classification text chosen by the user, initializing information concerning the cryptographic chaining (block or record) used in the file, an optional *user comment* field, and a cryptographic *key test* field. The key test field is based on the user key but does not contain it. During subsequent decipherment, IEBCODE can determine almost certainly, from the key test field, whether or not the key supplied at that time is the same as the key used in the original encipherment, thus providing early warning if the user has inadvertently supplied the wrong key. (The key test field consists of the XOR of the left and right halves of the encipherment of the ID record *time-and-date* field under the user key. Because of the XORing, the time-and-date and key test fields do not exhibit matching plaintext and ciphertext.)

When IEBCODE is used to *encipher* a file, it begins by initializing for encipherment under the user key and writing the encode ID record into the output file. Then each logical record in the input file is read and enciphered separately, and its ciphertext is transmitted to the output file.

When IEBCODE is used to *decipher* a file, it begins by searching for an encode ID record at the beginning of the file to determine the chaining mode and test the key. If a recognizable encode ID record is not found, all the data in the file is treated as ciphertext, and the ID record processing is bypassed. The ciphertext logical records are then read from the file one at a time and deciphered, and the plaintext equivalents are transmitted to the output file.

A cryptographic "round trip" using IEBCODE, from a plaintext file to a ciphertext file and, in a separate run, from ciphertext to plaintext again, naturally yields the original plaintext logical records. However, the data set blocking can be changed if requested by the user. And, as mentioned above, the method of encipherment is compatible with record-by-record cryptographic processing by user application programs.

IEBCODE control statement

A control statement is required for the user to specify the function to be performed and to supply required and optional parameters. At the user's option, the text of the control statement can be printed, or the printing can be bypassed for security reasons. The control statement options are:

- Function: Either ENCODE or DECODE must be specified.
- Cryptographic key: A key must be specified, but a wide variety of key formats is permitted, such as direct DES keys (expressed in hexadecimal) or long-character-string keys (expressed in either character or hexadecimal form). When hexadecimal notation is used, commas and blanks can be inserted

to improve readability without affecting the active key characters. If a DES key is supplied, it is used "as is" in the encipherment process. Longer keys are converted to DES keys by the chained encipherment method described above.

- Security classification: A parameter can be specified to place a classification text (in the clear) in the ID record.
- Cryptographic chaining: Either the block chaining or record chaining option can be chosen. If the option is not specified with ENCODE, record chaining is assumed.
- ID record comment: A comment of up to 40 bytes of text can be included in the ID record, if supplied on the control statement. This comment is not enciphered but remains as a single line of plaintext user documentation (of ownership, for example, or for file identification) in what otherwise (except for the security classification) is a file of unrecognizable ciphertext.

error conditions and messages

When an error occurs, IEBCODE attempts to continue execution, if it is reasonable to do so, but with an appropriate error message. For many conditions, execution must terminate; but the text of the message produced, and an expanded explanation in IPS user documentation, are intended to get the user past the difficulty in as short a time as possible.

IEBCODE can produce 49 separate informational, warning, or error messages. One reason for so many messages and conditions is that enciphered data sets are fundamentally different from other data sets in that they look like "garbage." When a conventional program produces incorrect output, it may be recognizable as such. With cryptographic programs, great care is needed to avoid producing files that look like garbage and really *are* garbage! (It is not unlikely that an installation will eventually use IEBCODE without testing each file of ciphertext to be certain it can be deciphered, in which case it is important that any errors be properly detected and reported.)

IEBCODE summary

IEBCODE processes a variety of sequential data sets and individual partitioned-data-set members. Because of this flexibility, users generally do not need to alter their data to use the IPS cryptographic system. IEBCODE's control statement is easily coded and provides a method for the simple entry of a cryptographic key in different formats, including character strings with meaning to individual users. The large number of error messages that can be produced under different circumstances, and their explanatory texts, help users solve difficulties quickly and easily.

IEBCODE is used to encipher files to be sent between computing centers and also to protect on-line data. For application programs that do not call IPS subroutines for encipherment and decipherment (see below), IEBCODE performs these services, typically with ciphertext in permanent data sets and plaintext in temporary ones.

IPS subroutines

Users are encouraged to call IPS subroutines from their own FORTRAN, PL/I Optimizing and Checkout Compiler, or Assembler programs. When calling IPS from an application program, the user passes data in main storage and receives data from IPS the same way. I/O is the user's responsibility and is handled in the usual way according to the programming language chosen. Since the subroutines depend on the chosen programming language for almost all operating system services (the major exception is the OS/VS TIME macro), we have found that they work equally well in any OS/VS system and in VM/370 CMS, and that ciphertext produced in any of these systems can be deciphered in any other.

One advantage of encipherment by a subroutine call, mentioned above under *The IEBCODE utility program*, is that no plaintext need ever exist on external files. Instead, the user program reads enciphered data from a file and deciphers it only at the time of use, and likewise enciphers data before writing it to a file. Therefore no plaintext data need ever exist outside of main storage (except for transient residence in paging data sets).

IPS subroutines can also be used for enciphering and deciphering nonsequential files, which cannot be processed by IEBCODE. Of particular interest are nonsequential files from which only selected records are read at any one time. The use of IPS subroutines avoids the cost of deciphering an entire file in order to access only a few records. In such applications, block chaining generally is more appropriate than record chaining.

Encipherment by a subroutine call also allows users to select the data that is to be enciphered in a file. If only certain fields of each record are to be concealed, for example customer name and address, those fields can be enciphered by a subroutine call, and the remainder of the record left in plaintext. This procedure can save a significant amount of CPU time if large files are being processed. If necessary, the various confidential parts of records can be enciphered under different keys, making it possible to grant selective access to the enciphered material according to the distribution of keys. In these cases also, block chaining is more appropriate than record chaining.

In general, the subroutines for different languages work in similar ways. Each set has four entry points: two for encipherment and two for decipherment. One of each pair is an initializer, to which the cryptographic key and the processing options are given. Initialization normally is performed once for each run. The other routine in each pair performs the actual encipherments or decipherments, one call for each record. We decided to separate the initialization function from the encipherment or decipherment function to save the time that otherwise would be required to initialize IPS working storage before processing each piece of data.

When the *encode* initializer is called, it is given the key, the security classification, and the cryptographic chaining option. The encode initializer prepares a table to speed the encipherments under the specified key, prepares for chaining, and passes back the encode ID record, which the user may optionally write into the output file to identify the data. The encode ID record is identical to that produced by IEBCODE. When the *decode* initializer is called, it is given the key and the optional encode ID record if saved at the time of encipherment, or the chaining information if needed because the ID record is missing. The decode initializer prepares the table for decipherment under the specified key and prepares for chaining according to the ID record or the user argument.

When the *encoder* is called, it is passed the plaintext and the area where it is to place the corresponding ciphertext. This area can be the plaintext area itself, since the routine can process data "in place." When the *decoder* is called, it is passed ciphertext and the area where it is to place the corresponding plaintext, which similarly can be the ciphertext area itself for "in place" decipherment.

The required attributes of the data to be enciphered are natural to the programming language the user has chosen, to the extent that we could arrange this. That is, in Assembler language, a user is expected to pass the addresses of the input and output data areas and their length. In FORTRAN, a user is expected to pass FORTRAN variables (usually arrays) for the data areas, and an integer variable containing their length (in bytes). In PL/I, a user is asked to pass either fixed or varying character strings, or a mixture. If PL/I data of other types is to be processed, the user is instructed to either pass BASED fixed-length character strings (whose pointers address the data to be processed), or define and initialize structures that imitate PL/I's character string descriptors, and call IPS through alternate entry points declared with OPTIONS (ASSEMBLER).

error handling

In the IPS subroutines, difficulties are divided into two classes, *warnings* and *errors*. And, like many other programs, the IPS subroutines take corrective action and continue, perhaps after a warning message, when a minor difficulty arises. But because there is a danger of transmitting wrong data in the belief that it is valid ciphertext, IPS terminates a user program in the presence of a serious error. Because termination is implemented with FOR-

TRAN and PL/I language features, not the system ABEND macro, users can easily override the termination if they choose. Termination occurs if all of the following three conditions hold:

- A difficulty defined as an IPS *error* has occurred—for example, the user has supplied a cryptographic key shorter than the DES minimum, or has requested that a negative number of data bytes be enciphered. (IPS *warnings*, resulting for example from a request to encipher a zero-byte record, do not cause termination.)
- The user has called IPS from a language for which the protection is defined; that is, PL/I in all cases, or FORTRAN if the installation's system programmers have prepared FORTRAN's extended error handling facility for use by IPS.
- The user has not prepared for the possibility of such errors; that is, he has not called the FORTRAN ERRSET routine, or not supplied a suitable PL/I ON-unit.

In other words, if a user is willing to handle errors, and signals that fact by establishing the correct FORTRAN or PL/I error environment, IPS will not terminate the program in the presence of an error. But if the user permits an error to occur and has not prepared for it, IPS will terminate the program to prevent destruction of the user's data.

We gave a great deal of thought to this problem before deciding to provide for the termination of user programs, but implemented it because we felt that users should be protected from serious errors. It would be possible for a user who supplies erroneous arguments to write either nonsense data or actual plaintext under the impression that it is valid ciphertext. If our indicative return codes were ignored in such situations, the result could be a serious loss, which we decided to guard against to the extent possible. To be sure, the protection is not perfect. It is not present in an Assembler language environment, and it is lost if the user takes the bypass actions offered in our user documentation. But these risks have to be taken if careful users are to be allowed to continue in execution.

Warning and error messages have as much variety and detail as those for the IEBCODE utility, but they are modified to report in terms of the programming language chosen. Assembler language users may choose from two versions of the IPS subroutines: one that presents only a return code, or one that presents both the return code and formatted error messages. FORTRAN and PL/I users will find messages in the standard output print file unless suppressed by standard language features (ERRSET in FORTRAN, ON-units in PL/I).

A limitation of the subroutines is their vulnerability to any user error in which the IPS tables are overlaid by user data. For example, except in PL/I with SUBSCRIPTRANGE enabled, it is possible to store data outside the boundaries of arrays. If such an operation should alter the IPS tables, the encipherment process would give incorrect results while appearing to be correct.

interactive IPS commands

Any programmer can encipher information interactively by including calls to IPS subroutines in his own interactive programs. In this sense, IPS has always been available to TSO and VM/370 application programmers. But for secretarial use, and for programmers who want a convenient way to encipher entire files, IPS includes a CIPHER command for VM/370 CMS files[13] and another for TSO sequential data sets.[20]

The VM/370 CMS CIPHER command is a *copy* utility combined with encipherment. With the *encode* option, CIPHER creates an enciphered copy of a CMS disk file. With the *decode* option, it creates a plaintext copy of a previously enciphered file. This command provides the easiest way to use IPS in the CMS environment. CIPHER is invoked either from a terminal or from a CMS EXEC file (one that contains a sequence of CMS commands to be executed). It is commonly used to encipher confidential documents that otherwise could not be stored on line, and to encipher confidential material to be sent from one location to another over the IBM VM/370 network.[22] The command itself initiates all terminal interaction and I/O, and it calls the IPS cryptographic subroutines for encipherments and decipherments. The ciphertext produced by CIPHER is IPS standard ciphertext. That is, each file of ciphertext contains an identifying record (the encode ID record) at the beginning, and each record in the CMS file is enciphered separately for convenient use by application programs. This format is the same as that used by the IEBCODE utility in OS/VS systems, and we have found that the OS/VS and CMS systems can exchange ciphertext without difficulty.

The TSO CIPHER command can be used in MVS[21] to encipher sequential data sets and individual partitioned-data-set members during a TSO session. CIPHER itself initiates all terminal interaction, then invokes IEBCODE for data set I/O and cryptographic services. Thus it ensures compatibility of ciphertexts between batch and interactive uses.

Applications of IPS

IPS first went into active use in November 1975 at the IBM Thomas J. Watson Research Center to encipher on-line data sets that contained confidential financial information. This data had been stored in plaintext on disk and was in constant use by a variety of reporting and analysis programs. The owners of the data were concerned about its vulnerability, but could not protect it except at an impractical cost in operational difficulties. (The use of locked-up disks and tapes, instructions to the machine operators to mount volumes only for jobs with certain names, and so on, were rejected.) The IPS utility IEBCODE was a welcome aid, because in one half-day of work, the owners were able to encipher the data, place deciphering job steps at the top of every job (IEBCODE wrote a plaintext temporary data set for the analysis programs that followed), and provide instruction for all concerned in using the utility.

IPS is available at more than one hundred IBM locations. A September 1977 user survey and personal communications revealed that IPS is used:

- As a component of one of IBM's internal telecommunications networks, for the automatic encipherment of very confidential data before transmission.
- To encipher and decipher APL workspaces stored as CMS files.
- To decipher tapes received from other IBM locations.
- To encipher VM/370 CMS SCRIPT files that contain documentation on, for example, unannounced program products or division operating plans.
- To encipher the contents of an OS password data set.
- To encipher the customer name and address field in records of installed IBM equipment, while leaving the remainder of the record in plaintext. This procedure saves CPU time and permits free use of the file for statistical analysis while protecting IBM's customer list.
- To encipher an on-line data base that contains information on the design of electronic circuit chips.
- To generate cryptographic keys by the repeated modification and encipherment of a random data string.
- To generate hash table values from long character strings, the hash value being the rightmost bytes of the block-chained encipherment of the character string.
- To store OS/VS load modules in enciphered form for protection against unauthorized execution. (The modules are first unloaded to a sequential form by a system utility program, then enciphered with IEBCODE.)
- To supplement the Resource Access Control Facility (RACF)[23] for data that must be kept secret even from those with RACF *special* or *operations* authority.

The major application of IPS appears to be the preparation of ciphertext files for transmission over IBM data networks. But as the above list shows, IPS is also used to protect on-line files, although in MVS much less frequently now than at first. Access control is less expensive than encipherment, and many MVS files are now protected by RACF rather than being enciphered by IPS; but some files are given both RACF and IPS protection, according to the importance of the data and the frequency with which it will be accessed. IPS continues to be used to protect on-line files in VM/370 CMS.

IPS and the IBM cryptographic products

Programmed Cryptographic Facility

The IPS cryptographic programs and the IBM cryptographic program products were developed separately for different purposes, and they have interesting similarities and differences. The IBM OS/VS1 and OS/VS2 MVS Programmed Cryptographic Facility[24] is a software implementation of DES, together with key management and handling services. Assembler language programmers can call on the Facility for the encipherment and decipherment of data in main storage. The Facility is used also by other IBM program products that require cryptographic services, such as the ACF/VTAM Encrypt/Decrypt Feature[25] for enciphered telecommunications and the Access Method Services (AMS) Cryptographic Option[26] for enciphered VSAM and nonVSAM data sets. Unlike IPS, the Programmed Cryptographic Facility will not run in VM/370 except as a component of an OS/VS1 or OS/VS2 virtual machine. The Facility uses the storage protection feature of IBM System/370 hardware, so its cryptographic tables cannot be damaged by the *store* error discussed above in our discussion of the IPS subroutines.

The IPS method of block chaining is also included in the IBM cryptographic products. Accordingly, Programmed Cryptographic Facility encipherments follow the IPS block chaining technique, with the initial chaining value supplied as a parameter at each call to the CIPHER macro. Record chaining is not now explicitly available, but it can be obtained by generating an output chaining value from each encipherment to be used as the input chaining value for the next encipherment, in the manner described earlier in this paper. With the correct setting of the initial chaining value, IPS and the Programmed Cryptographic Facility encipher and decipher data in an identical manner.

key management

We believe that requiring users to enter keys only at the time of their use for encipherment or decipherment and *not* storing them permanently in the system is an asset of IPS. But for many applications, cryptography would not be practical in computer systems if manual key entry were required. For instance, if a terminal can communicate with the host computer in ciphertext, the host must have the cryptographic key needed to start communications. There would be confusion and delay if each such key had to be entered manually by a system operator. Moreover, to prevent different sessions at a given terminal from using the same key, provision is made in the ACF/VTAM Encrypt/Decrypt Feature, in concert with the Programmed Cryptographic Facility, to generate cryptographic *session keys* by the system itself, rather than by a user. (The newly generated session key is transmitted to the terminal shortly after the user logs on, and it is enciphered under the terminal's own key. The terminal then switches keys for the rest of the session.)

For these and similar reasons, the Programmed Cryptographic Facility offers a key generation and management service that was not contemplated in the design of IPS. For file protection, users of the Facility can elect to supply their own private DES keys (IPS-like long-character-string keys are not accepted); they can request that the Facility generate keys as needed, and then treat those keys as private keys; or they can request key generation, then store enciphered versions of the keys with the associated data to permit automatic decipherment (that is, without the user having to supply the key) under various circumstances. This latter choice carries a risk that privileged users of the system might make unauthorized decipherments.

The greater the importance of keeping a particular file of data secure against a wide variety of threats, the more important it becomes to use private keys; but for many files the protection of encipherment is justified, while the inconvenience of private keys is not. IPS was designed in the belief that keys should not be stored in the system *in any form* and at a time when the protective features offered by MVS and used by the Programmed Cryptographic Facility were not available. Arguments can be advanced in favor of either the IPS or the Programmed Cryptographic Facility approach to key management, depending on different operating environments and attack scenarios. Under IPS, only private keys are allowed; the Programmed Cryptographic Facility gives the user a choice, which depends on the importance of the data, the relative safety of private keys, and the convenience of system-managed keys.

AMS Cryptographic Option

Another product, the Access Method Services (AMS) Cryptographic Option,[26] is roughly equivalent to the IPS IEBCODE utility as invoked through the TSO CIPHER command. Both are intended for the encipherment and decipherment of data files as a whole. The Cryptographic Option is an extension of the AMS REPRO command. As such, it can make enciphered copies of nonVSAM sequential data sets (as IEBCODE does) and process most VSAM files, which are inaccessible to IEBCODE. AMS does not perform encipherments directly, but calls on the Programmed Cryptographic Facility for the encipherment and decipherment of data, and for certain key handling services.

AMS-enciphered files by default use logical record encipherment with block chaining. To improve performance and short-record security, AMS can (at user option) read logical records in groups and encipher each group in one operation. This mode of processing resembles IPS record chaining because the chaining operation continues across record boundaries within the groups; but the method is not equivalent because DES eight-byte encipherment blocks are not necessarily aligned with logical record boundaries (as in IPS), and because the chaining operations are reset at the beginning of each group instead of continuing throughout the file. As a consequence, record grouping is probably not a good idea when user programs (not AMS itself) might be used to decipher such files.

key handling

AMS offers the key generation and key management services of the Programmed Cryptographic Facility. Therefore, unless the user enters a key for the encipherment of a file, one will be generated automatically. The generated key can be returned to the user in the clear, or it can be enciphered under another key, a *secondary file key*, and its encipherment returned to the user. Thus, access to both the enciphered text of the data key and to the secondary file key (by its identifying key name) is required to decipher the data file. At user option, the enciphered data key and the name of the secondary key can be placed in a file header to permit automatic decipherment under selected conditions.

The IBM 3848 Cryptographic Unit

The IBM 3848 Cryptographic Unit and the OS/VS2 MVS Cryptographic Unit Support Program[27,28] provide a hardware replacement for the software encipherments performed in the Programmed Cryptographic Facility. The 3848 attaches to an I/O channel and performs encipherments and decipherments of the data transmitted to it. The Cryptographic Unit Support Program replaces the Programmed Cryptographic Facility and transforms invocations of the CIPHER macro into the I/O operations required to drive the 3848.

The preceding discussion of the IPS cryptographic design and the Programmed Cryptographic Facility applies equally to the 3848 and its Support Program.

Where large amounts of data are to be enciphered or deciphered, the 3848 Cryptographic Unit should be considered, for by replacing the software encipherment routines in the Programmed Cryptographic Facility with hardware, significant savings of CPU time can be achieved.

Summary

The IPS programs have succeeded in bringing cryptography to a large community of users, both programmers and non-programmers, at IBM computing centers. IPS has contributed significant augmentations to the Data Encryption Standard: techniques of cryptographic chaining, which help to conceal the underlying structure of plaintext; a consistent definition of the encipherment of data of any length (not just multiples of eight bytes); and an increased number of ways in which cryptographic keys can be defined by users. Some of these implementation options are also included in the IBM cryptographic products.

ACKNOWLEDGMENTS

The authors especially thank Wayne Deniston, who implemented the VM/370 CIPHER command. Many others assisted in the design and development of IPS: Horst Feistel (now retired from IBM Research) designed the class of ciphers of which the Data Encryption Standard is an example; Carl Meyer and Walt Tuchman built upon Feistel's work and designed DES itself; Don Coppersmith worked on the original test implementations of the Feistel cipher; Andy Gibney implemented the TSO CIPHER command; David A. Smith and Leroy Junker gave valuable suggestions for the PL/I subroutine interface. Joseph Rogers and Richard Morron of Corporate Information Systems were helpful in introducing IPS to internal IBM computing centers.

Appendix: formal DES notation and description of chaining

The cryptographic function in IPS is provided by an augmentation of the DES *block cipher.* An *n-block* $\underline{x}$ is a sequence of $n \geq 1$ binary 0's or 1's

$$\underline{x} = (x_0, x_1, \cdots, x_{n-1}) \qquad x_i = 0 \text{ or } 1\ (0 \leq i < n)$$

Let $\{0,1\}^n$ denote the set of all n-blocks (for some *fixed* n)

$$\{0,1\}^n = \{\underline{x} = (x_0, x_1, \cdots, x_{n-1}) : x_i = 0 \text{ or } 1,\ 0 \leq i < n\}$$

A *block cipher* π on $\{0,1\}^n$ is a one-to-one transformation

$$\pi : \{0,1\}^n \rightarrow \{0,1\}^n$$

replacing each *plaintext* n-block $\underline{x}$ with a *ciphertext* n-block $\underline{y}$

$$\textit{plaintext} \quad \underline{x} \rightarrow \pi\{\underline{x}\} = \underline{y} \quad \textit{ciphertext}$$

Each n-block $\underline{y}$ is the ciphertext of a unique plaintext n-block $\underline{x}$. The inverse transformation, yielding plaintext from ciphertext, is denoted by π^{-1}. Note that the block cipher π replaces the block $\underline{x}$ of plaintext data by the block $\underline{y}$ of ciphertext data of the *same* length.

When a cryptographic transformation is employed by a pair of users to secure communications (by masking the content of messages), the users must incorporate some *secret* information in the cryptographic process so that the communication is intelligible only to them. The element of secrecy is provided by selecting one block cipher $\pi\{\underline{k}\}$ from a family of block ciphers which depend on a parameter $\underline{k}$, called a *key*. A *keyed block cipher* is a family of block ciphers

$$\pi\{\underline{k}\} : \{0,1\}^n \rightarrow \{0,1\}^n \qquad \underline{k} \in K$$

$$\underline{y} = \pi\{\underline{k},\underline{x}\}$$

where the key $\underline{k}$ is an element of some finite set K, called the *key space*. In any implementation of a cryptographic facility, each pair of users, who wish to communicate with enciphered messages, select in some manner an element $\underline{k}$ from K.

Desiderata of a good keyed block cipher are:

- The *input alphabet* $\{0,1\}^n$ should be "large";
- The key space K should be "large";
- The function $\pi\{\underline{k}\} : \underline{x} \rightarrow \underline{y}$, which relates the key, plaintext, and ciphertext, should be "complicated."

A "large" input alphabet effectively negates the possibility of an opponent's constructing a catalog of frequencies of plaintext-ciphertext pairs to which statistical analysis might be applied. A "large" key space effectively defeats exhaustive *key trial*. A "complicated" key-plaintext-ciphertext relation makes it difficult to detect any dependencies in the triple (key,plaintext,ciphertext) which would enable one to recover the key from corresponding plaintext and ciphertext, or recover plaintext from ciphertext.

Data Encryption Standard (DES)

DES is a keyed block cipher that transforms a 64-bit (eight-byte) plaintext block $(x_0, x_1, \cdots, x_{63})$ into a 64-bit ciphertext block $(y_0, y_1, \cdots, y_{63})$ under the control of a 56-bit key $(k_0, k_1, \cdots, k_{55})$. The key space $K = \{0,1\}^{56}$ contains 2^{56} different keys. There are 2^{64} possible plaintext 64-bit blocks $\{0,1\}^{64}$.

We use the notation DES$\{\underline{k},\underline{x}\}$ to denote encipherment of the plaintext block $\underline{x}$ by DES using the key $\underline{k}$, and the notation DES^{-1} $\{\underline{k},\underline{y}\}$ to denote decipherment of the ciphertext block $\underline{y}$ by DES using the key $\underline{k}$. It is believed that the DES algorithm is a "good" keyed block cipher in the sense described above. No cryptanalysis of DES, obtaining plaintext from ciphertext *without* the key, is known to the authors.

The DES algorithm is defined *only* for plaintext data blocks $\underline{x}$ with a length of eight bytes. The definition can be extended to plaintext records of arbitrary length (in bytes) in several ways. First, consider the case in which $\langle \underline{x} \rangle$[29] is a plaintext record whose length (in bytes) is a multiple of eight. Then

$\langle \underline{x} \rangle = (\underline{x}_1, \underline{x}_2, \cdots, \underline{x}_m)$

$(\underline{x}_i = (x_{i,0}, x_{i,1}, \cdots, x_{i,63})$: a block of eight bytes, $1 \leq i \leq m)$

The obvious way to encipher a plaintext record of $8m$ bytes is by applying DES to each eight-byte block, as follows:

$\langle \underline{y} \rangle = (\underline{y}_1, \underline{y}_2, \cdots, \underline{y}_m)$

$\underline{y}_j = \text{DES}\{\underline{k}, \underline{x}_j\} \qquad 1 \leq j \leq m$

This natural use of DES is attractive because portions of a record (or file) can be deciphered without decipherment of the entire record (or file). It suffers from the fact that identical plaintext blocks are enciphered into identical ciphertext blocks (assuming the same key). Thus, files with significant repetitive patterns, such as Assembler language source files or digital representations of line drawings, yield ciphertext in which such patterns are somewhat replicated. This repetition does not imply a weakness in the algorithm, and it cannot be used to decipher an enciphered file; nevertheless, it is disconcerting and easily remedied.

chaining

In IPS, we have introduced the notion of *chaining*. Chaining masks repetitions in plaintext blocks. The *chained encipherment of* $\underline{x}$ by DES is defined as follows: Given an eight-byte *initial chaining value* $\underline{\text{ICV}}$, *key* $\underline{k}$, and *plaintext* $\langle \underline{x} \rangle = (\underline{x}_1, \underline{x}_2, \cdots, \underline{x}_m)$, we define the *chained encipherment* $\text{DESCH}\{\underline{k}, \langle \underline{x} \rangle\}$ of $\langle \underline{x} \rangle$ by

$\underline{y}_0 = \underline{\text{ICV}}$

$\underline{y}_j = \text{DES}\{\underline{k}, \underline{x}_j \oplus \underline{y}_{j-1}\} \quad 1 \leq j \leq m$

where $\oplus$ = XOR. Thus the first plaintext block $\underline{x}_1$ is XORed with the initial chaining value $\underline{\text{ICV}}$ and enciphered by DES to yield $\underline{y}_1$, and the jth plaintext block $\underline{x}_j$ is XORed with the $(j-1)$st eight-byte block of ciphertext $\underline{y}_{j-1}$ and enciphered by DES to yield $\underline{y}_j$. If we wish to show the dependence of $\text{DESCH}\{\underline{k}, \langle \underline{x} \rangle\}$ on the initial chaining value, we write $\text{DESCH}\{\underline{k}, \langle \underline{x} \rangle | \underline{\text{ICV}}\}$.

DESCH as defined above is a one-to-one transformation; that is, a block of $8m$ bytes of ciphertext can arise from only one $8m$-byte block of plaintext. Indeed, if $\langle \underline{y} \rangle$ is an $8m$-byte block of ciphertext—that is, if $\langle \underline{y} \rangle = (\underline{y}_1, \underline{y}_2, \cdots, \underline{y}_m)$—the rules for decipherment $\text{DESCH}^{-1}\{\underline{k}, \langle \underline{y} \rangle\} = \text{DESCH}^{-1}\{\underline{k}, \underline{y} | \underline{\text{ICV}}\}$ are

$\underline{y}_0 = \underline{\text{ICV}}$: *initial chaining value*

$\underline{x}_j = \text{DES}^{-1}\{\underline{k}, \underline{y}_j\} \oplus \underline{y}_{j-1} \qquad 1 \leq j \leq m$

Both encipherment and decipherment proceed from "left to right." Note that we can decipher the fragment $(\underline{y}_j, \underline{y}_{j+1}, \cdots, \underline{y}_m)$ from the key $\underline{k}$ and the previous ciphertext eight-byte block $\underline{y}_{j-1}$.

Chaining, as defined here, possesses an important *self-healing property*. Suppose that $\langle \underline{\mathbf{x}} \rangle = (\underline{\mathbf{x}}_1, \underline{\mathbf{x}}_2, \cdots, \underline{\mathbf{x}}_m)$ (the record of $8m$ blocks of plaintext) has been enciphered, with chaining, into ciphertext $\langle \underline{\mathbf{y}} \rangle = (\underline{\mathbf{y}}_1, \underline{\mathbf{y}}_2, \cdots, \underline{\mathbf{y}}_m)$, and that an error has been made (in storage, say) in the jth ciphertext block $\underline{\mathbf{y}}_j$, replacing it with $\underline{\mathbf{y}}'_j$. Thus when decipherment is attempted, the ciphertext is $\langle \underline{\mathbf{y}}' \rangle = (\underline{\mathbf{y}}'_1, \underline{\mathbf{y}}'_2, \cdots, \underline{\mathbf{y}}'_m)$, with

$$\underline{\mathbf{y}}'_i = \begin{cases} \underline{\mathbf{y}}_i & \text{if } i \neq j \\ \underline{\mathbf{y}}_j \oplus \underline{\mathbf{e}} \neq \underline{\mathbf{y}}_j & \text{if } i = j \end{cases}$$

Decipherment of $\underline{\mathbf{y}}'$ with initial chaining value $\underline{\text{ICV}} = \underline{\mathbf{y}}_0$ results in $\langle \underline{\mathbf{x}} \rangle = (\underline{\mathbf{x}}'_1, \cdots, \underline{\mathbf{x}}'_m)$, where

- The first $(j - 1)$ plaintext blocks are correctly obtained:

  $$\underline{\mathbf{x}}'_i = \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}'_i\} \oplus \underline{\mathbf{y}}'_{i-1} = \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}_i\} \oplus \underline{\mathbf{y}}_{i-1} = \underline{\mathbf{x}}_i$$

  $$1 \leq i < j$$

- The jth and $(j + 1)$st deciphered plaintext blocks contain errors:

  $$\underline{\mathbf{x}}'_j = \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}'_j\} \oplus \underline{\mathbf{y}}'_{j-1} = \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}_j \oplus \underline{\mathbf{e}}\} \oplus \underline{\mathbf{y}}_{j-1}$$

  $$\neq \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}_j\} \oplus \underline{\mathbf{y}}_{j-1} = \underline{\mathbf{x}}_j$$

  $$\underline{\mathbf{x}}'_{j+1} = \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}'_{j+1}\} \oplus \underline{\mathbf{y}}'_j = \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}_{j+1}\} \oplus \underline{\mathbf{y}}_j \oplus \underline{\mathbf{e}}$$

  $$\neq \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}_{j+1}\} \oplus \underline{\mathbf{y}}_j = \underline{\mathbf{x}}_{j+1}$$

  Note: The jth plaintext block is entirely incorrect because an incorrect block was subjected to DES decipherment; the $(j + 1)$st plaintext block is in error only in the bit positions corresponding to the errors in $\underline{\mathbf{y}}'_j$.

- The self-healing property yields the $(j + 2)$nd through mth plaintext blocks correctly:

  $$\underline{\mathbf{x}}'_i = \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}'_i\} \oplus \underline{\mathbf{y}}'_{i-1} = \text{DES}^{-1}\{\underline{\mathbf{k}}, \underline{\mathbf{y}}_i\} \oplus \underline{\mathbf{y}}_{i-1} = \underline{\mathbf{x}}_i$$

  $$j + 1 < i \leq m$$

Thus an error in the jth eight-byte block of ciphertext results (upon decipherment) in errors in the jth and $(j + 1)$st eight-byte blocks of plaintext. The error does not propagate further.

It remains to extend this notion of chaining to plaintext blocks whose length either is greater then eight bytes but not a multiple of eight bytes, or is less than eight bytes. Let $\langle \underline{\mathbf{x}} \rangle$ be a block of $8m + s$ bytes with $0 < s < 8$, $m \geq 1$. Then

$$\langle \underline{\mathbf{x}} \rangle = (\underline{\mathbf{x}}_1, \cdots, \underline{\mathbf{x}}_m, \underline{\mathbf{x}}_{m+1})$$

where $\underline{\mathbf{x}}_{m+1}$ is a *short block* of s bytes.

The chained encipherment $\langle \underline{y} \rangle = \text{DESCH}\{\underline{k},\langle \underline{x} \rangle\}$ is defined to be

$$\langle \underline{y} \rangle = (\underline{y}_1, \cdots, \underline{y}_m, \underline{y}_{m+1})$$

where

$$\underline{\text{ICV}} = \underline{y}_0 : \textit{initial chaining value}$$

$$\underline{y}_j = \text{DES}\{\underline{k},\underline{x}_j \oplus \underline{y}_{j-1}\} \qquad 1 \leq j \leq m$$

$$\underline{y}_{m+1} = \underline{x}_{m+1} \oplus \text{LEFT}_s[\text{DES}\{\underline{k},\underline{y}_m\}]$$

where $\text{LEFT}_s[\cdot]$ means the leftmost s bytes of $[\cdot]$. That is, the first *m full* blocks are enciphered with chaining as just described, and the $(m + 1)$st (short) block of ciphertext is the XOR of the short block of plaintext, of length s, with the left s bytes of the encipherment of $\underline{y}_m$ by DES. It is easy to verify that DES so extended is one-to-one; that is, the ciphertext uniquely determines the plaintext. Decipherment proceeds from left to right and includes an *encipherment* step to redetermine $\text{LEFT}_s[\text{DES}\{\underline{k},\underline{y}_m\}]$.

Finally, we must define the encipherment of a *short record*; that is, an s-byte block where s is less than 8. We define the IPS *block chaining* encipherment of the short record $\underline{x}$

$$\underline{x} = (x_0, x_1, \cdots, x_{s-1}) \qquad 1 \leq s < 8$$

by

$$\underline{y} = \underline{x} \oplus \text{LEFT}_s[\text{DES}\{\underline{k},\underline{\text{ICV}}\}]$$

That is, we encipher the initial chaining value $\underline{\text{ICV}}$, and XOR the leftmost s bytes with the plaintext $\underline{x}$.

Block chaining, as defined here, has two limitations: First, with respect to the short records of a file, it is weak, constituting an *interrupted Vernam system.*[18] From enough such enciphered short blocks, an analyst could recover the Vernam key and thus decipher the short blocks. Even a more elaborate cryptographic function, which depended however only on the given plaintext short block, the key, and the starting initial chaining value, would at best be a simple substitution on the set of possible short blocks (and under our assumptions would be length-preserving) and thus might be subject to cryptanalysis. With the availability of record chaining, described below, such an elaborate encipherment of short blocks was not thought warranted.

In addition, if a file contains two or more records that begin with identical fragments, including the first t blocks, then independent DES encipherments of those records begin with identical ciphertext fragments, through the first t blocks. This too presents an exposure, although a much lesser one. In IPS we remedy these possible defects by extending DES still further by introducing the notion of *record chaining*.

Suppose that a file F consists of a sequence of r records

$F : \langle \underline{x}^{(1)} \rangle, \langle \underline{x}^{(2)} \rangle, \cdots, \langle \underline{x}^{(r)} \rangle$

Let $\underline{\text{ICV}}$ be the initial chaining value. The IPS encipherment of F under record chaining

$\langle \underline{y}^{(1)} \rangle, \langle \underline{y}^{(2)} \rangle, \cdots, \langle \underline{y}^{(r)} \rangle$

is defined according to the principle that *the input chaining value for the record* $\underline{y}^{(i)}$ *is the most recent eight bytes of ciphertext (where the ciphertext is considered to be prefixed by the starting initial chaining value).* More precisely, if we introduce the notations

ith *record input chaining value* : $\underline{\text{ICV}}^{(i)}$ $\quad 1 \leq i \leq r$

ith *record output chaining value* : $\underline{\text{OCV}}^{(i)}$ $\quad 1 \leq i \leq r$

then the encipherment of F is defined by:

$\underline{\text{ICV}}^{(1)} = \underline{\text{ICV}}$ *(initial chaining value)*

$\underline{\text{ICV}}^{(i)} = \underline{\text{OCV}}^{(i-1)} \quad 1 < i \leq r$

$\langle \underline{y}^{(i)} \rangle = \text{DESCH}\{\underline{k}, \langle \underline{x}^{(i)} \rangle | \underline{\text{ICV}}^{(i)}\} \quad 1 \leq i \leq r$

$\underline{\text{OCV}}^{(i)} = \text{RIGHT8}[\text{ICV}^{(i)} \| \langle \underline{y}^{(i)} \rangle] \quad 1 \leq i \leq r$

where RIGHT8 means the right-most eight bytes. Thus the first record $\langle \underline{x}^{(1)} \rangle$ is enciphered by DES into $\langle \underline{y}^{(1)} \rangle$ using the initial chaining value $\underline{\text{ICV}}$, and the ith record $\langle \underline{x}^{(i)} \rangle$ is enciphered by DES into $\langle \underline{y}^{(i)} \rangle$ using the initial chaining value

$\underline{\text{ICV}}^{(i)} = \underline{\text{OCV}}^{(i-1)} \quad (1 < i \leq r)$

It is simple to verify that we can recover the ith plaintext record $\langle \underline{x}^{(i)} \rangle$ from the ith ciphertext record $\langle \underline{y}^{(i)} \rangle$, the key, and the $(i - 1)$st output chaining value $\underline{\text{OCV}}^{(i-1)}$ (the immediately preceding eight bytes of ciphertext).

Record chaining does for records, including short records, what chaining accomplishes for repeated eight-byte blocks and trailing short blocks. Thus, even if a file contains identical records, or short records, the ciphertext records are all distinct, equally strongly enciphered, and self-healing.

CITED REFERENCES AND NOTES

1. *IBM System/360 Operating System Introduction*, IBM Systems Library, order number GC28-6534, available through IBM branch offices.
2. *IBM Virtual Machine Facility/370 Introduction*, IBM Systems Library, order number GC20-1800, available through IBM branch offices.
3. C. R. Attanasio, P. W. Markstein, and R. J. Phillips, "Penetrating an operating system: a study of VM/370 integrity," *IBM Systems Journal* **15**, No. 1, 102-116 (1976).
4. Many cryptographic systems are not well designed because the ciphertext they generate can be analyzed, and the corresponding plaintext revealed, without knowledge of the key. No absolute guarantees can be given, but the

cryptographic system used by IPS appears to be adequately strong. As far as we know, no practical method of recovering plaintext from ciphertext without the key has been discovered. One can attempt to decipher a given file by trying all possible keys until the true one is found; but such attacks will not be feasible until very much faster computers are built. Moreover, the damage, if any, will be limited to the given file and other files enciphered with the same key.

5. *Data Encryption Standard*, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington, DC (January 1977).
6. H. Feistel, "Cryptography and computer privacy," *Scientific American* **228**, No. 5, 15-23 (May 1973).
7. H. Feistel, *Cryptographic Coding for Data-Bank Privacy*, IBM Research Report RC 2827 (March 1970).
8. H. Feistel, W. A. Notz, and J. L. Smith, "Some Cryptographic Techniques For Machine To Machine Data Communications," *Proceedings of the IEEE* **63**, No. 11, 1545-1553 (November 1975).
9. W. F. Ehrsam, S. M. Matyas, C. H. Meyer, and W. L. Tuchman, "A cryptographic key management scheme for implementing the Data Encryption Standard," *IBM Systems Journal* **17**, No. 2, 106-125 (1978).
10. S. M. Matyas and C. H. Meyer, "Generation, distribution, and installation of cryptographic keys," *IBM Systems Journal* **17**, No. 2, 126-137 (1978).
11. R. E. Lennon, "Cryptography architecture for information security," *IBM Systems Journal* **17**, No. 2, 138-150 (1978).
12. The term *OS/VS* is intended to mean either the OS/VS1 or OS/VS2 MVS control program.
13. *IBM Virtual Machine Facility/370 CMS User's Guide*, IBM Systems Library, order number GC20-1819, available through IBM branch offices.
14. *Planning for Enhanced VSAM under OS/VS*, IBM Systems Library, order number GC26-3842, available through IBM branch offices.
15. *OS/VS2 MVS Data Management Services Guide*, IBM Systems Library, order number GC26-3875, available through IBM branch offices.
16. XOR and $\oplus$ are notations for *Exclusive-OR*, which is defined in *IBM System/370 Principles of Operation*, IBM Systems Library, order number GA22-7000, available through IBM branch offices. The Exclusive-OR operation, known also as bitwise addition modulo 2, returns (in each bit position of the quantities being XORed) a 0 bit if the two argument bits are alike — 0 0 or 1 1 — and a 1 bit otherwise. The XOR of two bit strings B1 and B2 yields a bit string B3, which has a special property: the XOR of B3 with B1 yields B2, and the XOR of B3 with B2 yields B1. This property is exploited in IPS block chaining and is the reason that the effect of chaining can be removed during decipherment.
17. That is, the resulting ciphertexts are different except for a probability of approximately 2^{-64}, which we neglect. This is a special case of the following useful observations: Because of the great complexity of the function defined by DES, we can regard it as a key-dependent, pseudorandom, one-to-one function from input blocks to output blocks. Suppose $\underline{y}_0$ and $\underline{y}'_0$ are initial chaining values, $\underline{x}_1, \underline{x}_2, \cdots$ and $\underline{x}'_1, \underline{x}'_2, \cdots$ are sequences of plaintext blocks, and $\underline{y}_1, \underline{y}_2, \cdots$ and $\underline{y}'_1, \underline{y}'_2, \cdots$ are the corresponding sequences of ciphertext blocks resulting from encipherment under block chaining using the same key $\underline{k} = \underline{k}'$. Then given two ciphertext blocks $\underline{y}_i$ and $\underline{y}'_j$, if $i = j$ and all of the equations

    $$\underline{y}_0 \oplus \underline{x}_1 = \underline{y}'_0 \oplus \underline{x}'_1, \underline{x}_2 = \underline{x}'_2, \cdots, \underline{x}_i = \underline{x}'_i$$

    hold, then $\underline{y}_i = \underline{y}'_i$.

    If $i = j$ and just one of the above equations does not hold, then $\underline{y}_i \neq \underline{y}'_i$. Otherwise, if $i = j$ and at least two of the above equations are not satisfied, *or* if $i \neq j$, then $\underline{y}_i \neq \underline{y}'_j$ with probability $1 - 2^{-64}$. As a special case, if $\underline{y}_i, \underline{y}_j$ ($i \neq j$) are ciphertext blocks of a single chaining sequence, then $\underline{y}_i \neq \underline{y}_j$ with probability $1 - 2^{-64}$ (regardless of whether or not $\underline{x}_i = \underline{x}_j$).

18. The encipherment described in the text, of *short records* under *block-chaining-but-not-record-chaining*, is an interrupted Vernam cipher (on those short records only, not on any records of eight or more bytes). This means that the plaintext and a sufficient length of key are XORed together to produce the ciphertext (Vernam) and that the use of the key is interrupted; that is, it starts over at the beginning at irregular intervals—at the beginning of each short record. The actual Vernam key (of which at most seven bytes are ever used) is the encipherment of the initial chaining value. *This system is easily cryptanalyzed* on the basis of a moderate amount of ciphertext. Its use is *not* recommended, and as noted in the text, the possibility of its use is provided *only for a very restricted purpose*.
19. Under the process described for record chaining, if a record contains at least eight bytes, then its OCV consists of the last eight bytes of its ciphertext. These may be a final full block, or a short block preceded by part of the preceding full block. But if the record is a short record of $l < 8$ bytes, then its (eight-byte) OCV is composed of the last $8 - l$ bytes of its initial chaining value concatenated on the right by the l bytes of its ciphertext. Thus, if the preceding record was also short, the OCV may in turn contain part of the initial chaining value of the preceding record, and so on. With successive short records, more and more of any initial chaining value will be "pushed off" at the left by the ciphertext arriving on the right. The OCV will always contain at least one new byte of ciphertext and will virtually always change from one record to the next; and identical plaintexts at different points in the file will virtually never yield identical ciphertexts.
20. *OS/VS2 TSO Terminal User's Guide*, IBM Systems Library, order number GC28-0645, available through IBM branch offices.
21. *OS/VS2 MVS Overview*, IBM Systems Library, order number GC28-0984, available through IBM branch offices.
22. E. C. Hendricks and T. C. Hartmann, "Evolution of a virtual machine subsystem," *IBM Systems Journal* **18,** No. 1, 111-142 (1979).
23. *OS/VS2 MVS Resource Access Control Facility (RACF) General Information Manual*, IBM Systems Library, order number GC28-0722, available through IBM branch offices.
24. *Programmed Cryptographic Facility Program Product General Information Manual*, IBM Systems Library, order number GC28-0942, available through IBM branch offices.
25. *IBM Cryptographic Subsystem Concepts and Facilities*, IBM Systems Library, order number GC22-9063, available through IBM branch offices.
26. *OS/VS1 and OS/VS2 MVS Access Method Services Cryptographic Option*, IBM Systems Library, order number SC26-3916, available through IBM branch offices.
27. *IBM 3848 Cryptographic Unit Product Description and Operating Procedures*, IBM Systems Library, order number GC22-7073, available through IBM branch offices.
28. *OS/VS2 MVS Cryptographic Unit Support General Information Manual*, IBM Systems Library, order number GC28-1015, available through IBM branch offices.
29. For consistency we adhere to the following notational conventions: $\underline{x}$ denotes a block of eight bytes unless otherwise stated; $\langle \underline{x} \rangle$ denotes a *record* consisting of an arbitrary number of bytes; and $\langle \underline{x}^{(1)} \rangle, \langle \underline{x}^{(2)} \rangle, \cdots, \langle \underline{x}^{(r)} \rangle$ denotes a *file* composed of the records $\{\langle \underline{x}^{(i)} \rangle : 1 \leq i \leq r\}$. (It would be possible in a similar fashion to extend the definitions to blocks and records of arbitrary lengths in *bits*, but we have no need for this in practice.)

The authors are located at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

Reprint Order No. G321-5125.