AAEC/E525

AAEC/E525

AUSTRALIAN ATOMIC ENERGY COMMISSION

RESEARCH ESTABLISHMENT

LUCAS HEIGHTS

A SOFTWARE IMPLEMENTATION OF THE KNAPSACK PUBLIC KEY CRYPTOGRAPHIC SYSTEM

by

M.R. DAVIDSON
R.P. BACKSTROM

December 1981

ISBN 0 642 59724 3

AUSTRALIAN ATOMIC ENERGY COMMISSION

LUCAS HEIGHTS RESEARCH LABORATORIES

A SOFTWARE IMPLEMENTATION OF THE KNAPSACK PUBLIC KEY CRYPTOGRAPHIC SYSTEM

by

M.R. DAVIDSON

R.P. BACKSTROM

ABSTRACT

A software implementation of the Merkle and Hellman public key crytographic system is described. The corresponding subroutines KEYGEN, ENCRYPT, DECRYPT are designed for convenient use on the AAEC central computer and provide the means for more secure disk storage of data.

National Library of Australia card number and ISBN 0 642 59724 3

CONTENTS

1. INTRODUCTION

With the proliferation of electronic communications systems and the growing use of computer data banks, the question of data security has become one of paramount importance to the private individual, the business corporation and the government alike. Data encryption schemes render stored data unintelligible to unauthorised persons, thereby providing protection against illegal, often remote, access to those data (whether deliberate or accidental) via a publicly accessible communications link. Modern cryptographic systems and their application in secure communications are discussed in detail in reviews by Lempel [1979], Simmons [1979], Popek and Kline [1979], and by Diffie and Hellman [1979] who also present a useful bibliography.

Originally, the security of ciphers depended on the secrecy of the entire process but later ciphers were developed for which the encrypting algorithm could be safely revealed; only the key which enabled correct enciphering and deciphering needed to be kept secret. An example of a conventional modern scheme is the Data Encryption Standard (DES) [National Bureau of Standards 1977] which was developed by IBM and is based on iterated substitution and permutation.

A new and exciting class of encryption methods now presented in the literature is that of public key cryptography using trapdoor one-way functions. In these methods, there are separate keys for encryption and decryption, and not only the algorithms but also the encrypting key can be revealed without endangering the security of the method. The encrypting key can be stored on disk for convenience or, indeed, can be published in a

directory like a telephone number. The decrypting key is kept secret. It is then computationally unfeasible for an unauthorised person to decrypt the data; the calculation would require perhaps thousands of years on the most powerful computer (i.e. encryption is a one-way function). However, if one knows the decrypting key, the calculation is simple (i.e. the one-way function has a trapdoor).

The concept of public key cryptography was first introduced by Diffie and Hellman [1976]. Since then various implementations have been published (e.g. Merkle and Hellman 1978; Rivest et al. 1978) together with accounts in the popular literature [Kolata 1977; Gardner 1977; Hellman 1979]. This report describes the software implementation, on the AAEC's IBM computing system, of the Merkle and Hellman [1978] public key cryptographic scheme.

2. THE MERKLE AND HELLMAN (MH) SCHEME

In this section we describe the MH cryptographic system. It is based on the knapsack problem; i.e. given a knapsack and a set of n rods of the same diameter as the knapsack and of lengths $a_1, a_2, \ldots, a_n$ find a subset of the rods that, if possible, completely fills the knapsack. That is, if $S = \sum_{i=1}^{n} a_i x_i$, where $x_i = 0$ or 1 and the sum S and the knapsack vector $\underset{\sim}{a} = (a_1, a_2, \ldots, a_n)$ are known, find a vector $\underset{\sim}{x} = (x_1, x_2, \ldots, x_n)$ if one exists. This belongs to the NP (non-deterministic, polynomial time) class of problems for which all known general methods of solution require calculation times which increase exponentially with size n. The best published algorithm [Horowitz and Sahni 1974] for solving this general knapsack problem requires

of the order of $2^{n/2}$ operations and $2^{n/2}$ words of memory. When n = 128 this corresponds to 7.2×10^{16} operations which would take about 2300 years on a computer which performs one operation every microsecond. However, the computational complexity of NP problems is measured by the difficulty in solving the worst case; special cases can exist for which the solution is easy. Nevertheless, it is believed that, for an arbitrary choice of knapsack vector $\underset{\sim}{a}$ and sum S, the problem is computationally impossible to solve when n is large (e.g. $n > 100$), although given a possible solution vector $\underset{\sim}{x}$, one can rapidly check it by doing the summation. In the context of encryption, vector $\underset{\sim}{x}$ is the bit pattern representing a block of the data to be encrypted and the knapsack vector $\underset{\sim}{a}$ is the public key. Encryption transforms the plaintext data block into ciphertext by forming the sum S. For large n, encryption remains a fast process but decryption (deriving $\underset{\sim}{x}$ from S and $\underset{\sim}{a}$) is, in general, computationally impossible.

If an arbitrary choice of public key were made, no one would be able to decode the encrypted data, not even the owner of the data. Some structure must therefore be hidden in the knapsack vector to enable a privileged person to decode the data quickly using a shortcut method. However, to the uninformed observer, the vector $\underset{\sim}{a}$ must continue to look like a random n-tuple.

Merkle and Hellman begin by considering knapsack vectors $\underset{\sim}{a}' = (a'_1, a'_2, \ldots, a'_n)$ where $a'_i > \sum_{j=1}^{i-1} a'_j$ for $2 \leq i \leq n$. Such a sequence in which each member is greater than the sum of all preceding members is described as 'superincreasing'. Knapsack problems based on superincreasing sequences are easy to solve by successive subtractions. If $S' = \sum_{i=1}^{n} a'_i x_i$ and

x_i is either 0 or 1, then $x_n = 1$ if and only if $S' \geq a'_n$, and for $i = n-1, n-2, \ldots, 1$, $x_i = 1$ if and only if $S' - \sum_{j=i+1}^{n} a'_j x_j \geq a'_i$. This simple knapsack vector $\underset{\sim}{a}'$ is then disguised as a trapdoor knapsack vector $\underset{\sim}{a}$ (to be used as the public key) by setting $a_i = w\, a'_i \pmod m$ where m and w are two large numbers such that $m > \sum_{i=1}^{n} a'_i$ and gcd* $(w,m) = 1$. The $\{a_i\}$ now incorporate a hidden structure while appearing as a pseudo-random sequence. A solution of the knapsack problem involving $\underset{\sim}{a}$ is believed to be impractical without a knowledge of m and w. Alternatively, knowing m and w (together these form the decrypting key) allows the quick conversion of the apparently difficult knapsack problem $S = \sum_{i=1}^{n} a_i x_i$ to the easily solvable problem $S' = \sum_{i=1}^{n} a'_i x_i$ (since gcd $(w,m) = 1$, there exists a w^{-1} such that

$$w^{-1} w = 1 \pmod m \text{ and we can compute}$$

$$S' = w^{-1} S \pmod m = \sum_{i=1}^{n} a'_i x_i \pmod m,$$

a relation which holds in integer as well as modulo arithmetic, since $m > \sum_{i=1}^{n} a'_i$).

* greatest common divisor

3. THE CHOICE OF KNAPSACK PARAMETERS

Since we are implementing this scheme on a computer with a 32-bit word, it is convenient to represent large integers (e.g. a_i, w, m) in base 2^{15} arithmetic. Each such integer is stored as an array of halfwords (15 bits plus a sign bit) and routines which perform arithmetic operations on these integers (see Knuth [1971] for algorithms) can use full words as accumulators. Below we choose specific parameters for our knapsack system in the spirit of Merkle and Hellman but make some additional concessions to economy of calculation time and data expansion.

The knapsack vector is taken to have size n = 128 (i.e. data are subdivided into blocks of length 16 bytes), and each a'_i has the form

$$(2^{i-1} - 1).2^{15} + r_0,$$

where r_0 is a uniformly distributed random integer in the interval $[1, 2^{15} - 1]$. The modulus m has the form

$$(2^{15} - 1).2^{135} + \sum_{j=0}^{8} r_j \, 2^{15j}$$

where r_j is uniform in the range $[1, 2^{15} - 1]$, and the multiplier w is derived from the form $\sum_{j=0}^{9} s_j \, 2^{15j}$, which is divided by its gcd with m to give w. The random integer s_9 is uniform on $[1, 2^{15} - 2]$, thereby ensuring that $w < m$, and the remaining s_j are uniform on $[1, 2^{15} - 1]$.

Note that each a'_i possesses 15 random bits whereas, in the example given by Merkle and Hellman, it has 100 random bits. This is, in practice, unlikely to affect the security of the system since the multiplier w and the modulus m used to form the new knapsack component a_i have 150 and 135 random bits respectively.

Parameters w and m and each element a_i of the public key vector are typically 150 bits long. After encryption, a data block of length n = 128 bits (16 bytes) can expand to 157 bits which is stored as a 21-byte array. This represents a 1.3:1 data expansion.

4. THE RANDOM NUMBER GENERATOR

A sequence of uniformly distributed pseudo-random numbers may be generated using the expression

$$R_{i+1} = \lambda R_i \pmod{P} \quad ,$$

provided λ and P are carefully chosen. This is the multiplicative congruential method [Knuth 1971]. McGrath and Irving [1973] note that $\lambda = 5^{15}$ ($= 71AFD498D_{16}$) and $P = 2^{47}$ is a good choice of parameter values, both theoretically [Coveyou and MacPherson 1967] and in practice (i.e. in extensive use in Monte Carlo simulations on a CDC computer at Oak Ridge National Laboratory).

Here we use a random number generator, written for IBM computers and originating from ORNL, in which λ is chosen to equal $1AFD498D_{16}$ (i.e. the leading 3 bits of 5^{15} are ignored so that the number conveniently fits into a 32-bit word) and $P = 2^{48}$ (48-bit arithmetic is used). A starting number for the pseudo-random sequence, based on the time of day clock which is updated every microsecond, is incorporated locally. The order of the 64-bit number (the last 12 bits are always zero) produced by the STORE CLOCK instruction is reversed and a starting number is formed from the 48 most rapidly changing bits. This process ensures that the most significant digits of that number have the greatest uncertainty.

5. THE ENCRYPTION/DECRYPTION SOFTWARE

The sofware implementation of the MH cryptographic system is divided into three distinct programs. They are the key generation program (KEYGEN), the encryption program (ENCRYPT) and the decryption program (DECRYPT). Three computer procedures, of the same names, have also been set up to simplify the job control language (JCL) required.

The original programs were written in FORTRAN, by the first author to establish the feasibility of the algorithms. They gave in-core encryption and decryption times, on the IBM3031 computer, of 4.3 CPU seconds and 10.9 CPU seconds per 100 cards respectively. These figures do not include I/O processing time, the time required to decompose the input for encryption, nor the time required to repack the output into bytes of plaintext.

We, therefore, decided to transcribe these programs from FORTRAN into IBM3031 assembler language with a view to reducing their execution times. They now stand at 1.5 CPU seconds and 4.1 CPU seconds per 100 cards respectively. This includes all I/O processing time, as well as internal data packing and unpacking times.

The public key (knapsack vector) is generated by KEYGEN and stored on disk for use by both the encryption and decryption programs. ENCRYPT uses only the public key and so may be run without loss of security as a normal batch jobstep on the IBM3031 computer. However, since both KEYGEN and DECRYPT require the private key to be given, they must be run interactively. It is not possible to maintain security and allow this sensitive key to be provided as normal job input because system programmers may be able to examine it directly from the system spool dataset. This means that KEYGEN and DECRYPT must be run as jobsteps of interactive CLASS=I jobs. It is not necessary to secure the programs KEYGEN, ENCRYPT or DECRYPT against public scrutiny; only the user's private key needs to be kept secret.

5.1 KEYGEN

The key generation process requires either a sequential dataset or a member of a partitioned dataset for storage of the public key data. The only restriction on this dataset is that it be defined by a logical record length (LRECL) of 80 bytes. The blocksize, however, may be chosen as any convenient multiple of 80 (such as 2560 or 6160). This allows the key to be stored as a member of a plaintext, partitioned dataset, and requiring space equivalent to 33 card images. The program may be invoked as follows:

```
//KEY1  EXEC KEYGEN,KEY='USR.KEY'
```

or

```
//KEY2  EXEC KEYGEN,KEY='USR.KEYLIB(KEY1)'
```

for a sequential or partitioned dataset (member KEY1) respectively.

The user must then establish an interactive session with KEYGEN via the Dataway computer link [Ellis 1970] using $RUN [Backstrom 1979]. The initial output informs the user that the terminal must be in upper/lower case mode because the private key produced is a 50-character upper and lower case string. To reduce the effort of memorising and re-entering such a large string, the user may choose to be told only the first N characters, provided that N is at least 10 and at most 50. The private key, with random characters in the first N places, is then stored on the public key dataset; the first N characters are restored when the user enters the correct values. The user may, therefore, request a 10-character decryption key and trade security for convenience, or retain up to 50 characters for ultimate protection.

The internally generated private key is really a pair of interleaved 150-bit numbers (w^{-1} and m) which are together specified by 91 decimal digits. Since terminal input is limited to 80 characters, this would have required a double read. However, using a 64-character alphabet (each character covering 6 bits), 300 bits of information require at most 50 characters to specify. The 64-character alphabet chosen consists of the 52 alphabetic characters a-z and A-Z, the ten digits 0-9 and the two characters $ and #.

After the keylength (10-50 characters) is chosen, the private key is displayed on the user's terminal three times to guard against data transmission errors. The user must then type back the given key to verify

receipt of the data. The key is split into five-character groups separated by commas to aid in the recording and entering of the value. If errors are made, KEYGEN will indicate the position and value of the error and again request verification. The user may abandon these attempts at any time by replying '/' to the key request, in which case KEYGEN will terminate with a condition code of 16. Note that the user is not permitted to specify the private key since it must be based on w and m which are required to satisfy certain specific conditions.

5.2 ENCRYPT

The encryption process needs an input and an output dataset as well as the public encryption key dataset. It does not require the private key, and so may be invoked as a normal jobstep on the IBM3031 computer. Because of the relatively high execution times required for encryption and decryption, the option is given to encrypt only specified ranges of a dataset. For example, if only lines 10-20, line 800 and lines 900-1000 need encryption, the rest can simply be copied to the output dataset. These ranges are specified in the PARM field as PARM='10-20,800,900-1000' . The input and output datasets must be specified as INPUT='Dataset Name' and OUTPUT='Dataset Name' on the EXEC card. Both datasets must have LRECL=80 bytes which allows the encrypted output to be stored in a partitioned dataset, along with normal plaintext members. Here again, the blocksize may be any convenient multiple of 80. The encryption process expands the output in the ratio 21:16 but the extra data are 'wrapped-around' onto subsequent records. If a substantial portion of a dataset is to be encrypted, allowance for the above expansion should be made when the output dataset is being allocated. To encrypt the entire dataset, specify: PARM='*' on the EXEC card.

Examples of the use of the procedure ENCRYPT follow:

```
//ENCR1 EXEC ENCRYPT,KEY='USR.KEYLIB(KEY1)',
//           INPUT='USR.SRCE',
//           OUTPUT='USR.OUT',
//           PARM='10-20,800,900-1000'

//ENCR2 EXEC ENCRYPT,KEY='USR.KEY',
//           INPUT='USR.LIBRARY(MEMB1)',
//           OUTPUT='USR.ENCR(MEMB1)',
//           PARM='1-20,500-510'

//ENCR3 EXEC ENCRYPT,KEY='USR.SOURCE(KEY1)',
//           INPUT='USR.SOURCE(SRCE1)',
//           OUTPUT='USR.SOURCE(ENCR1)',
//           PARM='*'
```

We thought it far preferable to use LRECL=80 for both the key dataset and the output of ENCRYPT than to insist on, say, LRECL=20 and 105 which are the internal logical record lengths. If the user were to use normal plaintext, partitioned datasets for the output from either program, the IBM operating system would have altered the LRECL of the dataset to 20 or 105 bytes respectively. This would have meant that subsequent access to the plaintext members would fail, necessitating system programmer advice and intervention to correct the situation.

Alternative specifications for the input dataset may be made by omitting the INPUT keyword on the EXEC card and adding an overriding DD-card for SYSUT1 describing the dataset. For example,

```
//SYSUT1 DD DSN=INPUTDSN,UNIT=(TAPE,,DEFER),
//          DCB=(RECFM=FB,BLKSIZE=6160,LRECL=80),
//          VOL=SER=USRTPE,LABEL=(1,NL)
```

for plaintext stored on an unlabelled magnetic tape. Plaintext which is generated by a user program may be written to a virtual input output (VIO) dataset, rather than a normal user dataset on disk or tape. The VIO dataset is then passed to the ENCRYPT step. In that case, the SYSUT1 DD-card must refer back to the VIO dataset generated in the previous jobstep. For example,

```
// EXEC FORTGCLG
...
generate plaintext and write to logical unit 9
...
//GO.FT09F001 DD DSN=&&TEMP,UNIT=VIO,DISP=(NEW,PASS),
//               DCB=(RECFM=FB,BLKSIZE=6160,LRECL=80),
//               SPACE=(TRK,(10,10))
//ENCR EXEC ENCRYPT,KEY='USR.KEYLIB(KEY1)',
//          OUTPUT='USR.SOURCE(ENCR1)',
//          PARM='10-20,500-600'
//SYSUT1 DD DSN=&&TEMP,DISP=(OLD,DELETE)
```

5.3 DECRYPT

The decryption process requires an input and an output dataset, as well as the public encryption key dataset. NOTE: OUTPUT from ENCRYPT is INPUT to DECRYPT. The private decryption key is also required, and so DECRYPT must be run as a jobstep of an interactive CLASS=I job. The input and output datasets must both have LRECL=80 and the decryption ranges must be specified via the PARM field in exactly the same way as for ENCRYPT.

The DECRYPT procedure may be invoked as follows:

```
//DECR1 EXEC DECRYPT,KEY='USR.KEYLIB(KEY1)',
//           INPUT='USR.OUT',
//           OUTPUT='USR.PLAINTXT',
//           PARM='10-20,800,900-1000'

//DECR2 EXEC DECRYPT,KEY='USR.KEY',
//           INPUT='USR.ENCR(MEMB1)',
//           OUTPUT='USR.DECR(MEMB1)',
//           PARM='1-20,500-510'

//DECR3 EXEC DECRYPT,KEY='USR.SOURCE(KEY1)',
//           INPUT='USR.SOURCE(ENCR1)',
//           OUTPUT='USR.SOURCE(DECR1)',
//           PARM='*'
```

An interactive session via the Dataway using $RUN should then be established by the user who should also ensure that the terminal is in upper/lower case mode. DECRYPT then requests the private key which should be entered in five-character groups, separated by commas, as given by KEYGEN. The key input is requested by DECRYPT via a 'silent read' command to maintain security. On some Dataway terminals, each character entered under this command will echo as an asterisk. This will give the user a positive check on the number of characters received by DECRYPT.

The user-supplied portion of the private decryption key is then combined with that stored on disk and the trapdoor calculation applied to the public encryption coefficients. If they are converted to the expected 'simple knapsack' form (as described previously), the key is accepted. If this fails (or the given key is not between 10 and 50 characters in length), the request is repeated at most four times (with a warning message before the final key request). This will discourage anyone from systematically trying billions of key combinations.

As soon as the private key is validated, the interactive segment of DECRYPT is ended with the message END OF INTERACTION. The program, however, continues the decryption and/or copying of records as specified in the PARM field. A subsequent jobstep then typically reads and processes the generated plaintext from the DECRYPT output dataset.

As for the input dataset to ENCRYPT, alternative specifications may be made for the output dataset to DECRYPT. In that case, the user should omit the OUTPUT keyword on the EXEC card and add an overriding DD-card for SYSUT2 describing the dataset. In particular, if further processing of the decrypted

data is required, a VIO dataset should be used to pass the plaintext to a subsequent jobstep. For example,

```
//DECR   EXEC DECRYPT,KEY='USR.KEYLIB(KEY1)',
//            INPUT='USR.SOURCE(ENCR1)',
//            PARM='10-20,500-600'
//SYSUT2 DD   DSN=&&TEMP,UNIT=VIO,DISP=(NEW,PASS),
//            DCB=(RECFM=FB,BLKSIZE=6160,LRECL=80),
//            SPACE=(TRK,(10,10))
...

...
//PGM    EXEC USERPROC,COND=(0,NE,DECR.DECRYPT)
//SYSIN   DD  DSN=&&TEMP,DISP=(OLD,DELETE)
...
```

5.3.1 Error messages from DECRYPT

The condition codes returned from DECRYPT indicate the success or otherwise of the program, with a return code of zero indicating no error. In the example above, the USERPROC procedure is not executed unless the step DECR passes a return code of zero. This precaution is worth noting so that CPU time is not wasted if the decryption fails. Other possible return codes and their meanings are listed below:

RETURN CODE	MEANING
4	Missing PARM field
8	Unknown character (not 0-9, -, * or comma)
12	Comma expected
16	Operands out of order
20	Unable to open SYSPRINT
24	Unable to open key dataset
28	Unable to open input dataset
32	Unable to open output dataset
36	LRECL not 80 for key dataset
40	LRECL not 80 for input dataset
44	LRECL not 80 for output dataset
48	Key dataset contains less than 33 records
52	DECRYPT cancelled by user (/ to key request)
56	DECRYPT cancelled (too many key retries)
60	Unexpected End of File during decryption

5.4 Encryption/Decryption Under TSO

KEYGEN, ENCRYPT and DECRYPT may also be called interactively under IBM's Time Sharing Option (TSO). The program $TSO may be used to log on to TSO from any user terminal on the Dataway. The three programs may be invoked using the command lists (CLISTS) KEYGEN, ENCRYPT and DECRYPT respectively as follows:

```
%KEYGEN   KEYDSN
%ENCRYPT  KEYDSN   INPUTDSN   OUTPUTDSN
%DECRYPT  KEYDSN   INPUTDSN   OUTPUTDSN
```

where the operands are self-explanatory. The PARM fields for ENCRYPT and DECRYPT are then requested as input from the terminal by the programs themselves. This is necessary since a PARM field containing commas cannot be passed to a program from an operand on a TSO CLIST because numbers following the first comma are rejected as exceeding the required count of CLIST operands. If INPUTDSN for ENCRYPT or OUTPUTDSN for DECRYPT is replaced by *, then plaintext may be entered or received, respectively, at the terminal. Similarly, but less usefully, ciphertext may also be channelled via the terminal.

The disadvantage of running DECRYPT under TSO, compared with a batch job, is that the output cannot be channelled usefully into a VIO dataset. This is because two different DD-names cannot refer simultaneously to the same VIO dataset under TSO. This, therefore, prevents subsequent user programs from reading the dataset back via arbitrarily chosen DD-names such as SYSIN, FT01F001 etc.

6. DATA SECURITY

There is no proof that any of the cryptographic schemes in current use (DES and the new public key methods) are as hard to break as they are supposed to be. In particular, although it is not computationally feasible to solve the general knapsack problem when n = 128, there is no guarantee that a method

cannot be discovered for breaking the cryptosystem with its built-in trapdoor without solving the corresponding general problem. To date, no weaknesses of the knapsack scheme have been demonstrated which do not involve exposure of trapdoor information. Unsubstantiated claims of successful breaking of public key schemes have been made by Herlestam [1978]. However, much of that paper has been shown to be erroneous [Rivest 1979]. The only serious study of the security of the MH scheme known to the authors has been made by Shamir and Zippel [1980] who show that, when m is known, an attempt to break the standard MH system will most likely be successful. They also note that this threat can be removed with the use of iterated knapsacks [Merkle and Hellman 1978]. Of course, the simplest answer is to ensure that private key information (of which the value of m forms a part) remains secret.

When the chosen private key length N = 10, there are 64^{10} (or about 10^{18}) unknown key combinations. However, an opponent need only search over the 10^9 unknown possibilities for m by repeatedly applying Shamir and Zippel's method of attack. This would take about 11 days of continuous computation assuming that the analysis of each possibility requires only one millisecond (a gross underestimate on the IBM3031). When N is increased to 50, that estimate rises to about 10^{34} years, and the opponent would be better off attacking the corresponding general knapsack problem.

Each bit of the ciphertext is an involved function of all the bits of the plaintext and the public key. If encryption produces ciphertext whose bit values are apparently random, then changing even a single bit of the plaintext should cause approximately 50 per cent of the ciphertext bits to change. Calculations confirm that this condition is well satisfied for the knapsack scheme used here. Random data blocks were generated and encrypted; in each

case the consequence of changing each of the data bits singly was observed and the results averaged. For 100 random data blocks, block averages of the percentage of significant ciphertext bits changed varied from 48.3 to 51.7 with an overall average of 49.7.

Often plaintext contains sufficient regularity (e.g. long strings of blanks or the use of key words) for a rough geometric outline between blank and non-blank characters to be detected in the ciphertext or for the recognition of ciphertext as program or data. Although this phenomenon may be intellectually disturbing it does not constitute a threat to security in the sense that it cannot assist an opponent in the search for the decrypting key or usable plaintext. Although such regularity can be removed by a process called 'chaining' [Konheim et al. 1980] no attempt is made to do so in this work.

Clearly, the overall security of any system is determined by its weakest link. In particular, although the cryptographic scheme used here appears to be highly secure, this is of no value if an opponent obtains the private key by tapping the communication link between user and central computer, or views the plaintext while it is located on disk before encryption or after decryption. In practice, the latter is a far more serious problem, since eavesdropping on terminal user - central computer transactions in this case does not seem likely owing to the risk of physical detection and the difficulty in identifying target information from other traffic on the communication line.

If plaintext is routed via a card reader or line printer, it may be examined conveniently on the system spool dataset by system programmers. This method of data transmission is, therefore, not recommended. However, if it must be used, the objective should be to minimise plaintext exposure by choosing job parameters that give the speediest job throughput. Where possible, plaintext data transfer between user and central computer should be made via terminal or physically secure magnetic tape.

It has been recommended that the passing of plaintext from one jobstep to another (e.g. from DECRYPT to a subsequent user program or from a user program to ENCRYPT) be accomplished by using a VIO dataset. A VIO dataset is created by the Multiple Virtual Storage (MVS) operating system on the IBM3031 and is similar to a temporary dataset with the advantage that no other user may view its contents. It is not written to disk like normal datasets, but may be 'paged' or 'swapped' in and out of memory to any of the 'page' datasets under the control of MVS. Thus a VIO dataset is as secure as any part of the user address space. Of course, system programmers will always be able to locate parts of a VIO dataset (albeit with difficulty), provided they are sufficiently motivated to determine the means and have the opportunity to implement it undetected. However, some risk of this type is inescapable with all operating systems, even MVS.

If normal user datasets are used for temporary storage of plaintext, the duration of such storage should be minimised by processing only small amounts of data at once and by erasing the contents of the datasets when they are no longer needed. On IBM systems, mere deletion of the dataset is not enough; the data are still on disk and a non-privileged user of the system can recover the original data, even accidentally. The only solution to this 'residual

data' problem is to destroy the contents of the dataset by overwriting it (e.g. with zeros).

7. CONCLUSIONS

A software implementation of the Merkle and Hellman [1978] public key cryptographic system, which emphasises economy of processing time, is described. The routines (KEYGEN, ENCRYPT, DECRYPT) have been packaged in convenient form for use on the AAEC central computer. They may be run either as normal IBM3031 jobsteps or from within a TSO session. The software provides the means for more secure disk storage of sensitive data, convenient processing of decrypted output and more secure transfer of data between users.

8. REFERENCES

Backstrom, R.P [1979] - Unpublished work.

Coveyou, R.R. and MacPherson, R.D. [1967] - Fourier analysis of uniform random number generators. J. ACM., 14:100-119.

Diffie, W. and Hellman, M.E. [1976] - New directions in cryptography. IEEE Trans. Inform. Theory, IT-22:644-654.

Diffie, W. and Hellman, M.E. [1979] - Privacy and authentication: an introduction to cryptography. Proc. IEEE, 67:397-427.

Ellis, P.J. [1970] - A multiplexed computer-computer, computer-device data link. AAEC/E206.

Gardner, M. [1977] - A new kind of cipher that would take millions of years to break. Scientific Amer., 237:(8)120-124.

Hellman, M.E. [1979] - The mathematics of public-key cryptography. Scientific Amer., 241:(2)130-139.

Herlestam, T. [1978] - Critical remarks on some public-key cryptosystems. BIT, 18:493-496.

Horowitz, E. and Sahni, S. [1974] - Computing partitions with applications to the knapsack problem. J. ACM, 21:277-292.

Knuth, D.E. [1971] - The Art of Computer Programming Vol.2 : Seminumerical Algorithms. Addison-Wesley, London.

Kolata, G.B. [1977] - Cryptography: on the brink of a revolution? Science, 197:747-748.

Konheim, A.G., Mack, M.H., McNeill, R.K., Tuckerman, B. and Waldbaum, G. [1980] - The IPS cryptographic programs. IBM Syst. J., 19:253-283.

Lempel, A. [1979] - Cryptology in transition. Comput. Surv., 11:285-303.

McGrath, E.J. and Irving, D.C. [1973] - Techniques for efficient Monte Carlo simulation, Vol.II. Report AD762722, Office of Naval Research, Arlington, Virginia.

Merkle, R.C. and Hellman, M.E. [1978] - Hiding information and signatures in trapdoor knapsacks. IEEE Trans. Inform. Theory, IT-24:525-530.

National Bureau of Standards [1977] - Data Encryption Standard. Federal Information Processing Standard (FIPS) Publication No.46, US Dept. of Commerce, Washington, DC., January.

Popek, G.J. and Kline, C.S. [1979] - Encryption and secure computer networks. Comput. Surv., 11:332-356.

Rivest, R.L., Shamir, A. and Adleman, L. [1978] - A method for obtaining digital signatures and public-key cryptosystems. Comm. ACM, 21:120-126.

Rivest, R.L. [1979] - Critical remarks on "Critical remarks on some public-key cryptosystems" by T. Herlestam. BIT, 19:274-275.

Shamir, A. and Zippel, R.E. [1980] - On the security of the Merkle-Hellman cryptographic scheme. IEEE Trans. Inform. Theory, IT-26:339-340.

Simmons, G.J. [1979] - Symmetric and asymmetric encryption. Comput. Surv., 11:305-330.