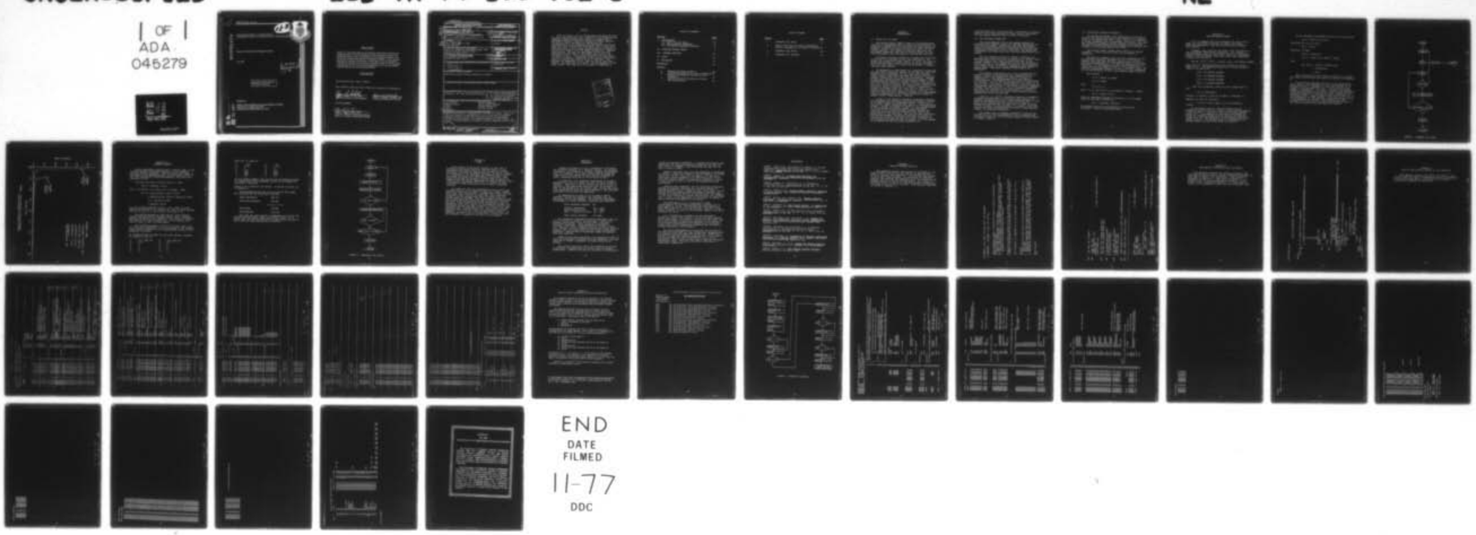
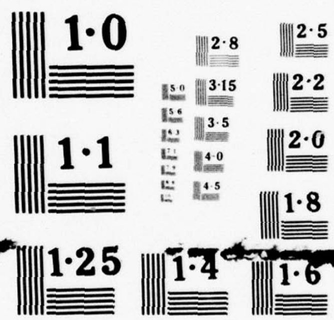
AD-A045 279 ELECTRONIC SYSTEMS DIV HANSCOM AFB MASS F/G 9/2
MULTICS SECURITY EVALUATION: PASSWORD AND FILE ENCRYPTION TECHN--ETC(U)
JUN 77 P J DOWNEY
UNCLASSIFIED ESD-TR-74-193-VOL-3 NL

1 OF 1
ADA
045279



END
DATE
FILMED
11-77
DDC


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5

ESD-TR-74-193, Vol. III

MULTICS SECURITY EVALUATION: PASSWORD AND FILE ENCRYPTION TECHNIQUES

Deputy for Command and Management Systems

June 1977

Approved for Public Release;
Distribution Unlimited.

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
HANSCOM AIR FORCE BASE, MA 01731

LEGAL NOTICE

OTHER NOTICES

This technical report has been reviewed and is approved for publication.

ROGER R. SCHELL, Lt Col, USAF
ADP System Security Program Manager

BRIAN W. WOODRUFF, Captain, USAF
Techniques Engineering Division

FOR THE COMMANDER

FRANK J. EMMA, Colonel, USAF
Director, Computer Systems Engineering
Deputy for Command & Management Systems

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ESD-TR-74-193, Vol III-3

2. GOVT ACCESSION NO.:

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): MULTICS SECURITY EVALUATION: PASSWORD AND FILE ENCRYPTION TECHNIQUES.

5. TYPE OF REPORT & PERIOD COVERED: Final Report, March 1972 — June 1973

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Peter J. Downey, 1 Lt, USAF

8. CONTRACT OR GRANT NUMBER(s): In-House

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Deputy for Command and Management Systems (MCI), Electronic Systems Division (AFSC), Hanscom AFB, Mass. 01731

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Program Element 64708F, Project 6917

11. CONTROLLING OFFICE NAME AND ADDRESS: Hq. Electronic Systems Division, Hanscom AFB, Mass. 01731

12. REPORT DATE: June 1977

13. NUMBER OF PAGES: 41

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 45p.

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report):

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):

18. SUPPLEMENTARY NOTES: This is Volume III of a 4 Volume report: Multics Security Evaluation. The other volumes are entitled: Vol. I: Results and Recommendations; Vol. II: Vulnerability Analysis; Vol. IV: Exemplary Performance Under Demanding Workload

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):

Access Control
Computer Security
Multics
Privacy
Protection
Password Encryption
Secure Computer Systems
Security Penetration
Security Testing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

Passwords are stored in enciphered form in the Multics system. There is no clear text listing of the password file. In 1972, as part of a security analysis of Multics, an ESD team successfuly inverted the enciphering algorithm in use on Multics. This report documents the team's efforts. As a result of the ESD analysis, an improved encryption algorithm is now in use on Multics.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

127 100

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

PREFACE

This is Volume 3 of a 4 volume report prepared for the Air Force Data Services Center (AFDSC) by the Directorate of Computer Systems Engineering, Deputy for Command and Management Systems, Electronic Systems Division (ESD/MCI). The entire report represents an evaluation and recommendation of the Honeywell Multics system carried out under Air Force Project 6917 from March 1972 to June 1973. Work described in this volume was performed by personnel at ESD/MCI with support from the MITRE Corporation. Computer facilities at the Rome Air Development Center and the Massachusetts Institute of Technology were used in the evaluation effort. This volume was primarily authored by 1Lt Peter L. Downey. Additional inputs to the text made by James P. Anderson and Captain Brian W. Woodruff. The programs in Appendices B and C were written by Capt Paul Karger. The algorithm for "better" was developed by Lt Col Roger Schell, who also wrote the program in Appendix D.

TABLE OF CONTENTS

LIST OF FIGURES

SECTION I
INTRODUCTION

1.1 Basis for the Study

The Multics system <ORG71> began at the Massachusetts Institute of Technology (MIT) in 1964 with the objective of producing a computer utility embracing the whole complex of hardware, software and users to serve as a model for other similar systems. The impetus for Multics came from the successful Compatible Time Sharing System (CTSS) that had been developed at MIT previously on the IBM 709 and 7094.

The Multics system, because of its ambitious objectives of serving as a prototype utility was concerned at the outset with protection issues clearly in mind. The specific mechanisms designed to permit sharing of various objects (i.e. programs, data etc.) safely are discussed in <GRA68>, <SCH72a>, and <SCH72b>.

In response to a requirement for "advanced" interactive and multilevel processing imposed by the USAF and other DOD customers in the Pentagon, the Air Force Data Services Center (AFDSC) identified the Honeywell Multics system as a computer system that could meet the requirements. Because of the multi-(security)level processing ancitipated, AFDSC commissioned the Electronic Systems Division (ESD) of the Air Force Systems Command to conduct a security analysis of Multics to ascertain whether the system could indeed provide multilevel secure processing in a "benign" (restricted access) environment where all users have at least a Secret clearance and some users have a Top Secret clearance. As a result of ESD's security analysis, Honeywell has implemented security enhancements to the Multics operating system to support AFDSC's requirements. <WHI73>

As part of the security analysis, penetration attacks were organized and tested on the Multics systems at MIT and at the USAF's Rome Air Development Center. Volume II of this report <KAR74> describes the results of the penetration attacks. Volume III reports in detail on one accomplishment of the penetration exercise, namely, the successful inverting of the "non-invertible" password enciphering algorithm which was used in Multics at the time of the penetration in 1972 and 1973. The relative ease with which the Multics password enciphering algorithm was broken is instructive in showing the care with which a password or file enciphering algorithm must be chosen. It is an example of how easily one can be misled regarding the "non-invertibility" of an algorithm. The method of analysis exploits some simple methods of number theory, and points out the general approach taken in such an analysis.

1.2 Why Scrambled Passwords?

The login password file of any system presents an attractive and tempting target for would-be penetrators of time sharing systems. Such files have been the primary target of numerous penetration exercises, because with the information contained in them a penetrator can masquerade indefinitely and effectively for long term exploitation of a system.

For the reasons outlined in <KAR74>, obtaining the password file internally from the system is of minimal value to a penetrator who is also an authorized user of the system. However, for a penetrator who may not always be an authorized user of the system, obtaining a user's password is of great value. In addition, passwords might appear in memory dumps. Therefore, password files need to be protected.

The special vulnerability of a list of passwords and owners to attack by penetrators was recognized by R.M. Needham <WIL72> who proposed the idea of storing the ciphertext of an encrypted password with the owner's identification. He proposed that the cipher transformation be 'one-way', that is, for this particular use, there is no need to have a reversible transformation (the usual case for cryptographic applications). The reasoning behind Needham's proposal was that even if the file of encrypted passwords and their user identifiers was compromised, it would be impossible to ascertain the user's input password and thus masquerading would be prevented.

Evans et. al. <EVA74> elaborates somewhat on Needham's proposal and discusses, in a heuristic way, families of password scrambling functions. The interesting part of that paper is the observation that some of the primitive scrambling functions must be non-linear in order to defeat analytic attacks on the algorithms. In general, Evans has covered the major considerations involved in using one-way ciphers for password protection.

The scheme used to scramble passwords on Multics was devised prior to and independently of the considerations outlined by Evans (or in the related papers by Purdy <PUR74> or Johnson <JOH74>).

1.3 The Multics Password Scrambler

In the Multics system, user passwords are protected by storing the encrypted version of the password in a segment known as the Person Name Table (PNT). This is the only form in which a password list is maintained in Multics. No clear text listing of passwords exists anywhere in the system. The PNT is further protected from unauthorized access by the contents of its Access Control List (ACL).

The one-way cipher scheme used in Multics is called scramble_. A PL/1 listing of the routine appears in Appendix A.

The Multics scrambler works by first compressing the 8 Multics-ASCII character password from 72 to 56 bits by removing the high-order two bits (always zero in the 9 bit Multics representation of 7 bit ASCII characters) from each character. If the password is less than 8 characters in length, blanks were added to make it 8 characters long. The resulting compressed password, called p, is then multiplied by its own low-order 16 bits, then reduced modulo 10**19-1.

The notation

(1) $R = \mathrm{mod}(D, C)$ means

(2) $D = C*Q+R$

with

(3) $0 \leq R < C$

and C, D, Q, and R are all non-negative integers. Define

(4) $a = \mathrm{mod}(p, 2**16)$

Then the compressed password conversion to r, the number stored in the PNT, is given by

(5) $r = \mathrm{mod}(p*a, 10**19-1)$

Two attacks on this "non-invertible" function were developed. These are discussed below.

SECTION II
TRAILING BLANKS ATTACK

If it is assumed that most passwords are less than or equal to 6 non blank characters in length (the human lassitude hypothesis), they can be brute force decrypted very rapidly.

Scramble_ left justifies the ASCII input characters in the 8 character field before encryption. As a result, a password whose length is less than or equal to 6 characters contains trailing blanks (octal 040) which when compressed create a p of the form:

p=b(56), b(55), b(54).....b(18), b(17), XX 0100000 0100000

where the b(i) denotes arbitrary bits and each X can be either 0 or 1. On inspection there are only four possible lower 16 bit patterns:

a(1) = 00 0100000 0100000

a(2) = 01 0100000 0100000

a(3) = 10 0100000 0100000

a(4) = 11 0100000 0100000

From (2) we observe, where Q is the integer part of D/C,

(6) Q = floor(D/C).

Letting c = 10**19-1, from (5) we have r = mod(p*a, c).

Applying (1) and (2) we obtain

(7) p*a = c*q+r, where q is a non-negative integer.

In the special case of trailing blanks, we are attempting to find p in (7), given r (the encrypted value of p from the PNT), c (10**19-1), and the only four possible values of a. In attempting a brute force decryption by trying all possible values of q for each of the possible values of a, the only deterrent is the maximum value q can obtain from (7). The maximum value of q determines the maximum number of trials that would be required.

We can determine the maximum value of q by noting that

$$(8)\quad q = (p*a-r)/c \leq p*a/c$$

By definition of a, we have

$$(9)\quad a < 2**16$$

Similarly, we have

$$(10)\quad p < 2**56 \text{ and}$$

$$(11)\quad c < 2**64 \text{ (i.e. } 10**19 < 2**64\text{)}$$

then

$$(12)\quad p*a/c < (2**16)\ (2**56)/2**64$$

$$< 2**72/2**64$$

$$< 2**8$$

The significance of this result is that a is so small that only a little over 250 trials are required to determine p.

A brute-force algorithm unscr (r, a, p) was created which finds a valid password, p, which corresponds to the encrypted value, r, provided a = low order 16 bits of p. Figure 1 is a flowchart for unscr. (A listing for unscr appears in Appendix B). The unscr subroutine was applied to a PNT which contained 1082 entries. Unscr either printed a recovered password, or reported a failure (passwords > 6 characters long). Figure 2 depicts the cost in CPU time of this program on the 1082 entries. Sixty-two percent of all of the passwords on the MIT Multics system were thus obtained with little effort. The figure shows the cost in CPU time to recover short passwords was minimal.

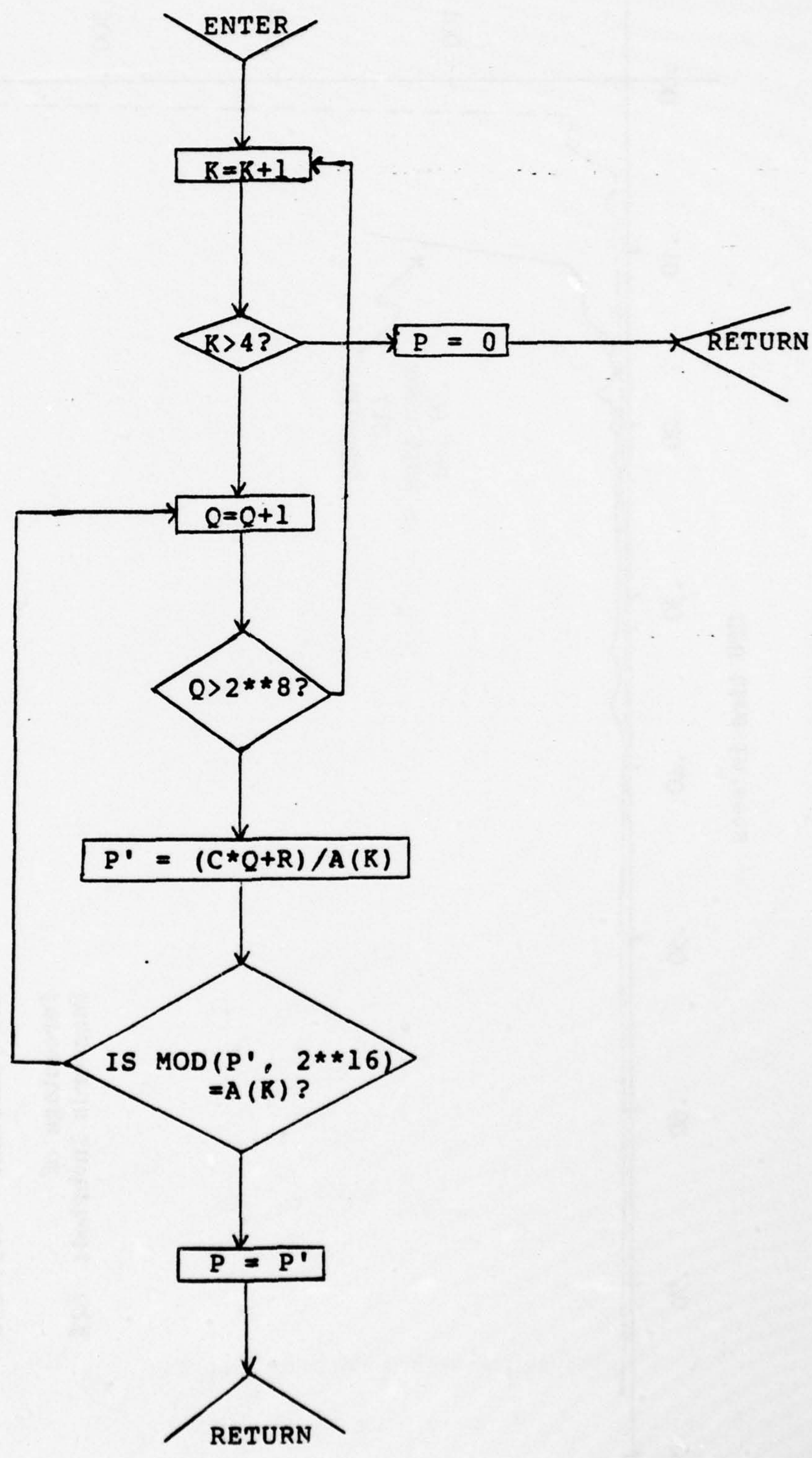
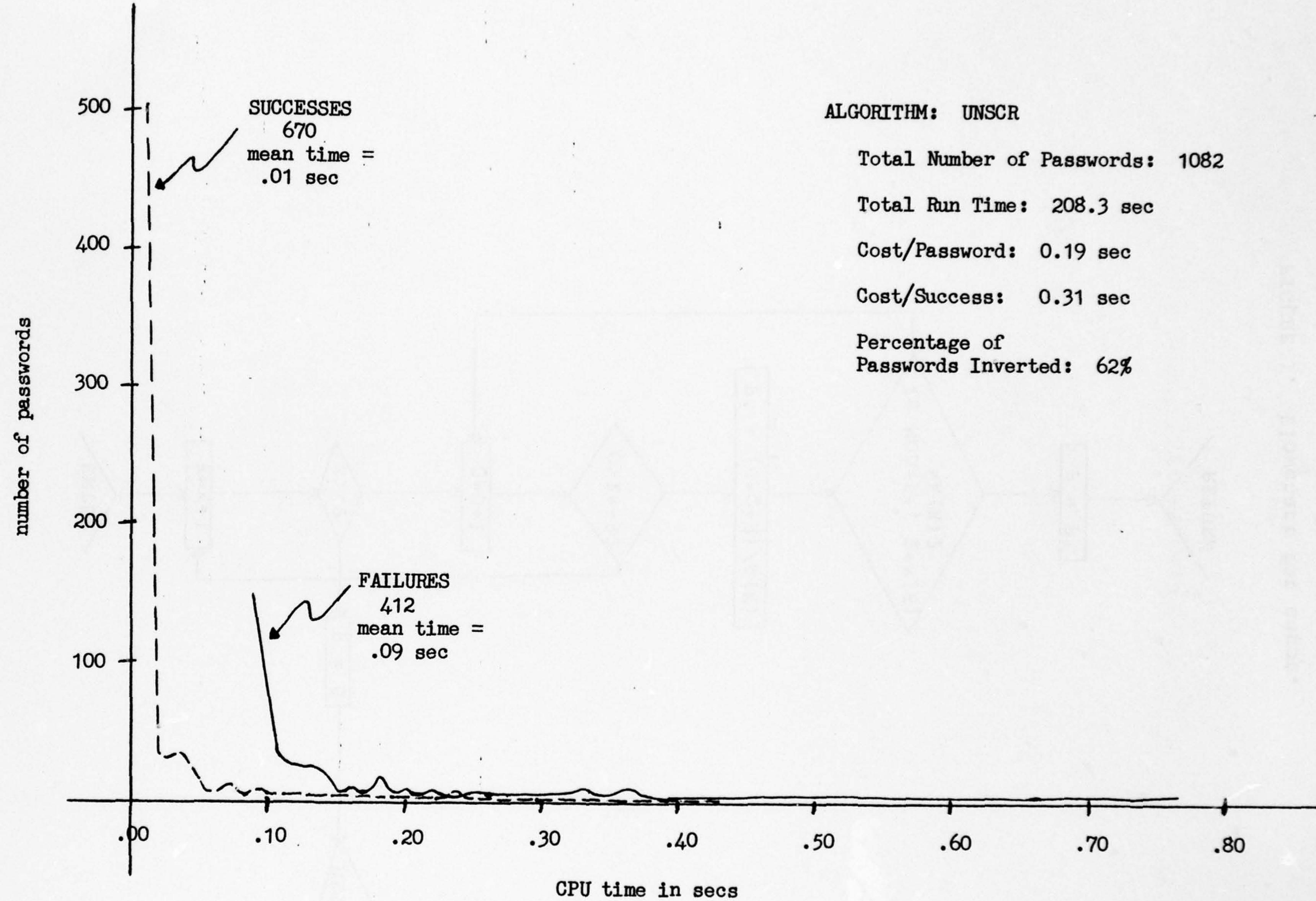


FIGURE 1. Flowchart for unscr.



ALGORITHM: UNSCR

Total Number of Passwords: 1082

Total Run Time: 208.3 sec

Cost/Password: 0.19 sec

Cost/Success: 0.31 sec

Percentage of
Passwords Inverted: 62%

FIGURE 2. Cost in CPU time to either successfully invert a password or to report a failure

SECTION III
GENERAL SOLUTION

Having developed a solution for a special case, it was of some interest to determine whether or not a general solution could be obtained. Such a solution was found; it was based on the observation that the low order 16 bits of p*a (the immediately transformed password (56 bits)) are identical to a*a.

Call the low order 16 bits of p*a, d. Then,

(13) d = mod(p*a, 2**16)

Let p = x*(2**16) + a, where x is an integer. Then,

(14) d = mod((x*2**16 + a)*a, 2**16)

= mod(x*a*2**16, 2**16) + mod(a*a, 2**16)

= 0 + mod(a*a, 2**16)

= mod(a*a, 2**16)

Let the function mod(a*a, 2**16) = g(a). Then let h(d) denote the inverse of the function g. That is, h(d) denotes the list of all of a, (a ≤ 2**16), with g(a) = d.

The general decryption algorithm was called better. Better first generates a two-part table. Part I contains possible values of d and a pointer. The pointer is either a null pointer (if h(d) is empty), or it is a pointer to a value of h(d) in part 2 of the table.

The interesting aspect of h(d) is the fact that it is quite sparse; consequently it has the potential for rapidly discriminating whether or not a hypothesized inversion (of a password) is correct.

To illustrate what is meant by h(d) being sparse, consider an example in base 10:

a	g(a) (mod 10)	a	g(a) (mod 10)
0	0	5	5
1	1	6	6
2	4	7	9
3	9	8	4
4	6	9	1

Then h(d) is given by:

d	h(d)	d	h(d)
0	0	5	0
1	1,9	6	4,6
2	null	7	null
3	null	8	null
4	2,8	9	3,7

In this simple example, only six out of ten possible values of d would need to be considered further in an attempt to unscramble a password.

Figure 3 is a flowchart for better. A listing of better is in Appendix C.

The procedure better was run on a PNT of 1085 names. The following CPU run times were recorded.

Table generation	200 sec
Inverting Passwords	148 sec
Total Time	348 sec
Cost/Password	0.32 sec

Note that the cost figure is comparable to that for the special case decryption based on trailing blanks. On a larger PNT, the average cost would be less since the cost of the d and h(d) table generation is constant.

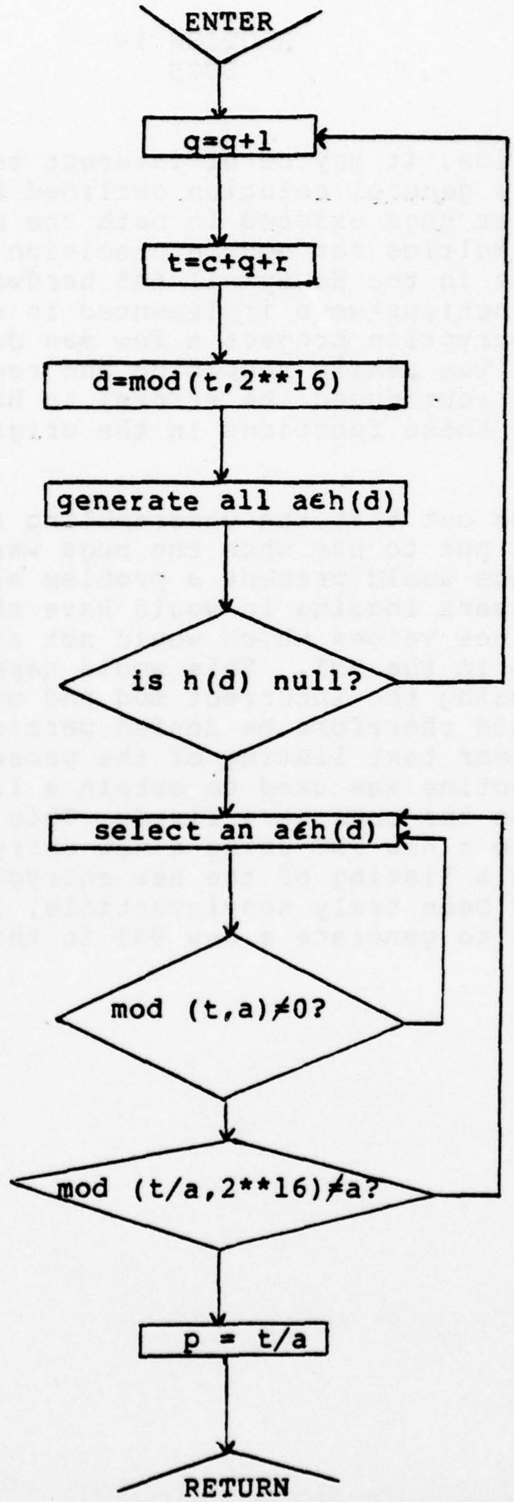


FIGURE 3. Flowchart for better.

SECTION IV
BUGS

As an aside, it may be of interest to note that in developing the general solution outlined above, it was discovered that bugs existed in both the mod and multiply functions in Multics for double precision integers. Because of limitations in the Honeywell 645 hardware instruction set, these functions were implemented in software. This caused the decryption project a few man days of effort to diagnose what was really happening and required special code (that exactly reproduced the errors) to handle the bugs introduced by these functions in the original scrambling function.

It turned out that the unscrambling routine which was developed was put to use when the bugs were fixed. Simply fixing the bugs would present a problem since after the bugs were fixed, users logging in would have their passwords scrambled to new values which would not agree with the values listed in the PNT. This would happen because the PNT was created using the incorrect mod and multiply functions. The users would therefore be denied permission to log in. Because no clear text listing of the passwords existed, the unscramble routine was used to obtain a listing of all user passwords when the bugs were fixed. This listing was then converted into a new PNT using a new encryption routine. Appendix D is a listing of the new encryption routine. If scramble_ had been truly non-invertible, it would not have been possible to generate a new PNT in this fashion.

SECTION V
CONCLUSION

Password encryption is not a fundamental requirement for providing security in a computer system. As discussed previously, if a penetrator is able to access the password file in a computer, he is most likely able to access any other file in that system as well, and the knowledge of users' passwords would be of little value to him.

In addition, it is generally unnecessary to access a password file from the system in order to obtain a user's password. Obtaining a user's password may be as simple as copying it from some place where the user has written it. Even if the password is not written down, it has been found that passwords can often be easily guessed if the users are permitted to pick their own passwords.

To demonstrate how easy it is to guess a user's password, the Multics password list obtained from the decryption effort was sampled. The following approximate percentages of "easily guessed" passwords were observed:

Total Passwords Sampled:	325
Directly Associable	100 - (31%)
Common English Words	50 - (15%)
Short (3 letters or less)	20 - (6%)
Total Easily Guessed:	170 (52%)

The category of passwords directly associable with the person covered two types of associations. Names, both personal and project, were one type used to provide associable passwords. These passwords consisted of initials, first names, reversed spelling, friend's names, etc. Numbers, such as telephone numbers and social security numbers, were the second type used as directly associable passwords. Combinations of associable names and numbers were also observed.

Based on this quick analysis, the conclusion is that if users are permitted to provide their own passwords, the work required to "guess" a password is highly likely to be minimal.

Not letting users pick their own passwords is one way to minimize the possibility of having a user's password compromised. <GAS75> describes an algorithm for generating random pronounceable passwords. By generating pronounceable passwords, the algorithm produces a password the user is more likely to remember, thus minimizing the need for the user to write it down.

Another technique using one-time passwords is described in <RIC73>. Under this scheme, a user's password is changed every time the user logs into the system. In this way, obtaining a user's password is of less value since it may have been changed before the penetrator attempts to log on. It also serves to alert a user if his password has been compromised.

Despite these drawbacks, use of a properly constructed one-way enciphering algorithm can provide a measure of additional security to a system at very little cost. It can provide security against anyone obtaining a password list through an accidental dump of the system files, for example. It will also discourage a system administrator from thinking that in order to be responsible to his duties, he needs to keep a listing of passwords in his office.

The development of an "irreversible" cipher transformation for encrypting passwords is harder than it would appear at first. The Multics algorithm appears to have been selected in an ad hoc fashion. Even so, to the casual observer, it would at first glance appear to be quite difficult to invert.

It is interesting to observe the two approaches embodied in <EVA74> and <PUR74>; one which creates complex ad hoc algorithms, the behavior of which are not known, the other which adopts an analytical approach to the design of the algorithm and computes the probability of successful attack under stated assumptions regarding what is known or assumed available to the attacker.

For future developments in this area, one or more of the functions discussed by Evans appears more promising than the function which had been used on Multics. ESD/MCI has provided an improved password scrambler that is now used in Multics although the "non-invertibility" of it is not guaranteed. This routine is also used in Multics to encrypt and decrypt files. The basic encryption algorithm is contained in Appendix D.

REFERENCES

<EVA74> Evans, A.J., Kantrowitz, W., Weiss, E., "A User Authentication Scheme Not Requiring Secrecy in the Computer", Communications of the ACM, Vol 17, No. 8, August 1974, pp 437-442.

<GAS75> Gasser, M., A Random Word Generator for Pronounceable Passwords, ESD-TR-75-97, November 1975 (AD A017676).

<GRA68> Graham, R., "Protection in an Information Processing Utility," Communications of the ACM, Vol 11, No. 4, May 1968 pp 365-369.

<JOH74> Johnson, S.M., Certain Number Theoretic Questions in Access Control, RAND Corporation Report R-1494-NSF, January 1974.

<KAR74> Karger, P.A., Schell, R.R., Multics Security Evaluation: Vulnerability Analysis, ESD-TR-74-193, Vol II, June 1974 (AD A001120).

<ORG71> Organick, E., The Multics System: An Examination of its Structure, MIT Press, Cambridge, Mass., 1971.

<PUR74> Purdy, C.B., "A High Security Log-in Procedure", Communications of the ACM, Vol 17, No. 8, August 1974, pp 442-445.

<RIC73> Richardson, M.H. and Potter, J.V., Design of a Magnetic Card Modifiable Credential System Demonstration, MCI-73-3, Electronics Systems Division, Hanscom AFB, December 1973.

<SCH72a> Schroeder, M. and Saltzer, J., "A Hardware Architecture for Implementing Protection Rings," Communications of the ACM, Vol 15, No. 3, March 1972, pp 157-170.

<SCH72b> Schroeder, M., Cooperation of Mutually Suspicious Subsystems in a Computer Utility, Ph.D. Thesis, MIT, 1972. (Also as MIT Project MAC Report TR-104.)

<WHI73> Whitmore, J., et al, Design for Multics Security Enhancements, ESD-TR-74-176, December 1973 (AD A030801).

<WIL72> Wilkes, M.V., Time Sharing Computer Systems, American Elsevier, New York, 1972.

APPENDIX A
Password Scramble Listing

This appendix contains the listing of "scramble_," the program used on Multics at the time of the ESD security study to encipher user passwords. This routine was invoked at every user login attempt. It enciphered the password typed at the terminal for comparison with the version of the password stored in the Person Name Table. As a result of the ESD security analysis, an improved password enciphering algorithm is now in use on Multics. This improved algorithm is listed in Appendix D.

```
scramble_: proc (arg) returns (char (8) aligned);

/* SCRAMBLE_ - Scramble a char (8) string.

   This procedure, given a password as input, returns an 8-character
   output string which:
   1. bears some relationship to the input
   2. loses some information - some passwords may scramble to the same value
   3. has no obvious relation to the input ("aaaaaaa" and "aaaaaab"
      scramble to noticeably different values.)

   Passwords stored in system files are scrambled, so that if anyone gets
   a dump of the password file by accident, it won't do him much good.

   The transform is supposed to be non-invertible. I am not sure it is.

   Method:
   1. strip the two high-order bits of each ASCII character, packing to
      the right.
   2. treat the resulting 56-bit quantity as an integer (note that it is
      positive). multiply this number by the low-order 16 bits of itself.
   3. divide the resulting product by 10**19-1 and return the remainder.

   THVV 10/30/71
    */
```

```
dcl   arg char (8) aligned;

dcl   temp char (8),
      templ fixed bin (71),
     (pl,p2) ptr,                                    /* ptrs to based overlays */
     (i,k) fixed bin;

dcl   bbt bit (72) aligned based (pl),
      bc8 char (8) aligned based (p2);

dcl  1 tsx based (p2) aligned,
     2 pad bit (16) unal,
     2 z (8) bit (7) unal;

dcl  1 tsy based (p2) aligned,
     2 pad bit (56) unal,
     2 b16 bit (16) unal;

dcl   const fixed bin (71) int static init (9999999999999999999);

dcl  (addr, fixed, mod, substr) builtin;

/* ---------------------------------------------------- */

      temp = arg;                                    /* copy argument */
      pl = addr (temp);
      p2 = addr (templ);
      templ = 0;
      k = l;
      do i = 3 to 72 by 9;
            z(k) = substr (bbt, i, 7);               /* squeeze out always-zero bits */
            k = k + l;
      end;
      templ = templ * fixed (b16,16);                /* sort of square the number */
      templ = mod (templ, const);
      return (bc8);

end;
```

APPENDIX B
Unscrambling Listing for Short Passwords

This appendix contains the listing of "unscr," the unscrambling routine used to invert enciphered passwords of less than or equal to six characters. This routine is discussed in section II. When unscr is applied to a password that has been enciphered by scramble_, it will either return a recovered password, or it will report a failure if the password is more than six characters long.

```
print unscr.pl1

                    unscr.pl1 10/24/72  1020.0 edt Tue


 unscr:
     proc (r, a, v) returns (bit (1) aligned);
          dcl
               c fixed bin (71) aligned init (9999999999999999999) int static,
             (     r,
                   a,
                   q,
                   t,
                   v,
                   w,
                   h)
               fixed bin (71),
               pp ptr;
          dcl
             (     sysprint,
                   sysin)
               file;
          dcl
               1 bits based (pp) aligned,
                    2 pad bit (56) unal,
                    2 b16 bit (16) unal;
          h = 10000000000000000b * (10000000000000000000000000000000000000000000b*a - a) + a*a;
          h = divide (h, c, 71, 0) + 1;
          do q = 0 to h;
               if mod (q, 1000) = 0 then put data (q);
               t = c*q +r;
               if mod (t, a) = 0 then
                    do;
                         v = divide (t, a, 71, 0);
                         w = v - a;
                         pp = addr (w);
                         if b16 = "0"b then return ("1"b);
                    end;
          end;
          return ("0"b);
     end unscr;
```

APPENDIX C
General Unscrambling Listing for All Passwords

This appendix contains the listing of "better", the routine which was used to successfully invert all passwords in the Person Name Table. The nature of the general solution is discussed in section III.

```
ASSEMBLY LISTING OF SEGMENT >udd>Multics>better.alm
ASSEMBLED ON:       12/22/72  0903.2 est Fri
OPTIONS USED:       list  symbols  new_call  new_object
ASSEMBLED BY:       ALM Version 4.1, November 1972
ASSEMBLER CREATED:  12/14/72  0612.4 est Thu

000000                                        name    better
000000                000000                  entry   better
                                              temp    lost_bit
                                              tempd   someplace
                                              tempd   dividend
                                              tempd   odiv
                                              tempd   linkptr
000000                            better:
000000  aa    000140 6270 00                  push
000001  aa  7 00040 2721 20
000002  aa  0 00002 2371 20                   ldaq    ap|2,*     "copy input argument
000003  aa  0 00006 3521 20                   eapbp   ap|6,*     "pointer to pointer to index table
000004  aa  2 00000 3521 20                   eapbp   bp|0,*     "pointer to index table in bp
000005  aa  0 00010 3701 20                   eaplp   ap|8,*     "pointer to pointer to value table
000006  aa  4 00000 3701 20                   eaplp   lp|0,*     "pointer to value table in lp
000007  aa  6 00060 6501 00                   stplp   linkptr    "saved value of value table ptr
000010  aa  6 00050 4501 00                   stz     lost_bit   "initialize to positive product
000011  aa    000022 7100 00                  tra     bl_entry

000012                            big_loop:
000012  aa  6 00056 2371 00                   ldaq    odiv
000013  0a    000146 0370 00                  adlaq   modulus
000014  0a    000022 6050 00                  tpl     bl_entry   "if no overflow to sign bit, keep trying
000015  aa  6 00050 2261 00                   ldx6    lost_bit   "else get pass indicator
000016  0a    000135 6010 00                  tnz     error_return        "if positive we've tried it all
000017  aa    400000 2660 03                  orx6    =o400000,du         "else mask for a negative product
000020  aa  6 00050 7461 00                   stx6    lost_bit   "to use in loops
000021  aa  0 00002 2371 20                   ldaq    ap|2,*     "and reinitialize adding loop
000022                            bl_entry:
000022  aa  6 00056 7571 00                   staq    odiv
000023  aa    216000 4110 03                  lde     =71b25,du  "normalizing exponent to e
000024  aa    400000 4750 03                  fad     =0.,du     "add zero to float odiv
000025  aa  6 00052 4571 00                   dfst    someplace  "floated version of dividend
000026  0a    000142 5670 00                  dfdv    fmod       "divide by floating version of 10**19-1
000027  0a    000140 4770 00                  dfad    k71b25     "add magic number to make integer
000030  0a    000142 4630 00                  dfmp    fmod       "multiply back to get mod
000031  aa    000000 5130 00                  fneg               "set to subtract from odiv
000032  aa  6 00052 4771 00                   dfad    someplace  "subtract
000033  aa    216000 4350 03                  ufa     =71b25,du  "fix remainder
000034  aa  0 00002 1171 20                   cmpaq   ap|2,*     "compare against input
000035  0a    000012 6010 00                  tnz     big_loop   "if different, no inner loop
000036  aa  6 00052 2351 00                   lda     someplace  "else get precision loss
000037  aa    000012 7710 00                  arl     10         "exponent to index alignment
000040  aa    777701 6240 01                  eax4    -63,au     "reduce by no-loss precision and put in x4
000041  0a    000043 6050 00                  tpl     set_ones   "and go mask dividend
000042  aa    000000 6240 00                  eax4    0          "no loss of precision
000043                            set_ones:
000043  aa  6 00056 2371 00                   ldaq    odiv
000044  0a    000161 2750 14                  orq     set_ones_table,4    "make lost bits in dividend ones
000045  aa  6 00050 2751 00                   ora     lost_bit   "restore lost bit from multiply
000046  aa  6 00054 7571 00                   staq    dividend
000047  aa    000000 6200 05                  eax0    0,al
000050  aa    000000 6210 06                  eax1    0,ql       "lower dividend in x0 and x1
000051  0a    000172 3700 14                  anq     mask_table,4        "adjust range of sqrt scan for precision loss
000052  0a    000161 7... 14                  ...     set_ones_table,4    "number of ... resulting from precision l
```

```
000053 aa   000000 6270 06          eax7    0,ql        "square in x7
000054 0a   000060 7100 00          tra     mid_loop_entry      "go start working on squares
000055                      mid_loop:
000055 aa   777777 6230 13          eax3    -1,3        "count squares processed
000056 0a   000012 6040 00          tmi     big_loop    "if done go to outer loop
000057 aa   000001 6270 17          eax7    1,7         "else do next square
000060                      mid_loop_entry:
000060 aa 2 00000 7251 17           lxl5    bp|0,7      "get count of roots for this square
000061 0a   000055 6000 00          tze     mid_loop    "if none get next square
000062 aa 6 00060 3701 20           epplp   linkptr,*   "get lp pointer to base of value segment
000063 aa 2 00000 2261 17           ldx6    bp|0,7      "get disp to first root
000064 aa 4 00000 3701 16           epplp   lp|0,6      "and pointer to it in lp

000065                      loop:
000065 aa   777777 6250 15          eax5    -1,5        "decrement square root count
000066 0a   000055 6040 00          tmi     mid_loop    "done when negative
000067 aa 6 00054 2361 00           ldq     dividend    "get first 18 bits of dividend
000070 aa   000022 7720 00          qrl     18
000071 aa 4 00000 5061 15           div     lp|0,5      "divide by trial root
000072 aa   000030 6360 10          eaq     0,0         "next 18 bits
000073 aa   000022 7330 00          lrs     18          "set up with remainder
000074 aa 4 00000 5061 15           div     lp|0,5
000075 aa 6 00055 2361 00           ldq     dividend+1          "get lower dividend upper half
000076 aa   000022 7330 00          lrs     18          "add previous remainder
000077 aa 4 00000 5061 15           div     lp|0,5
000100 aa   000000 6360 11          eaq     0,1         "get rest of lower dividend word
000101 aa   000022 7330 00          lrs     18
000102 aa 4 00000 5061 15           div     lp|0,5
000103 0a   000150 3150 14          cana    compare_table,4     "check for almost zero remainder
000104 0a   000065 6010 00          tnz     loop        "if non-zero, try again

000105 aa   000000 6260 06          eax6    0,ql        "save quotient in X6
000106 aa 4 00000 1361 15           sblq    lp|0,5
000107 aa   177777 3160 07          canq    65535,dl    "mod 2**16
000110 0a   000065 6010 00          tnz     loop        "try again if not 0 mod

000111 aa 6 00054 2361 00           ldq     dividend    "get quotient
000112 aa   000022 7720 00          qrl     18
000113 aa 4 00000 5061 15           div     lp|0,5
000114 aa   000000 6220 06          eax2    0,ql
000115 aa   777774 3020 03          canx2   =o777774,du
000116 0a   000065 6010 00          tnz     loop

000117 aa 0 00004 3521 20           eppbp   ap|4,*      "make bp point to output destination
000120 aa 2 00000 7421 00           stx2    bp|0
000121 aa   000000 6360 10          eaq     0,0
000122 aa   000022 7330 00          lrs     18
000123 aa 4 00000 5061 15           div     lp|0,5      "not previously
000124 aa   000000 6220 06          eax2    0,ql
000125 aa 2 00000 4421 00           sxl2    bp|0        "saved in loop
000126 aa 6 00055 2361 00           ldq     dividend+1          "and store
000127 aa   000022 7330 00          lrs     18
000130 aa 4 00000 5061 15           div     lp|0,5      "in bp|0
000131 aa   000000 6250 06          eax5    0,ql
000132 aa 2 00001 7451 00           stx5    bp|1        "return quotient
000133 aa 2 00001 4461 00           sxl6    bp|1

000134                      return:
000134 aa 7 00042 7101 20           return

000135                      error_return:
000135 0a   000144 2370 00          ldaq    minus_one
000136 aa 0 00006 7571 20           staq    ap|6,*
000137 0a   000134 7100 00          tra     return
```

```
                                        "
                                        "          DATA
                                        "
                                                   even
000140  aa   216000 000000     b71b25:    oct        216000000000,000000000000
000141  aa   000000 000000
000142  aa   200425 434430     fuol:      oct        200425434430,110474777777
000143  aa   110474 777777
000144                         minus_one:
000144  aa   777777 777777                dec        -1
000145  aa   777777 777777                dec        -1

000146                         modulus:   "(10**19)-1-1
000146  aa   001053 071060                oct        001053071060
000147  aa   221171 777776                oct        221171777776
000150                         compare_table:
000150  aa   777777 777777                oct        777777777777
000151  aa   777777 777776                oct        777777777776
000152  aa   777777 777774                oct        777777777774
000153  aa   777777 777770                oct        777777777770
000154  aa   777777 777760                oct        777777777760
000155  aa   777777 777740                oct        777777777740
000156  aa   777777 777700                oct        777777777700
000157  aa   777777 777600                oct        777777777600
000160  aa   777777 777400                oct        777777777400
000161                         set_ones_table:
000161  aa   000000 000000                oct        0
000162  aa   000000 000001                oct        1
000163  aa   000000 000003                oct        3
000164  aa   000000 000007                oct        7
000165  aa   000000 000017                oct        17
000166  aa   000000 000037                oct        37
000167  aa   000000 000077                oct        77
000170  aa   000000 000177                oct        177
000171  aa   000000 000377                oct        377
000172                         mask_table:
000172  aa   000000 177777                oct        177777
000173  aa   000000 177776                oct        177776
000174  aa   000000 177774                oct        177774
000175  aa   000000 177770                oct        177770
000176  aa   000000 177760                oct        177760
000177  aa   000000 177740                oct        177740
000200  aa   000000 177700                oct        177700
000201  aa   000000 177600                oct        177600
000202  aa   000000 177400                oct        177400

                                          end
```

26

```
INTERNAL SEQUENCES

000203  5a   000010 0000 00
000204  aa   7 00046 2721 20
000205  0a   000000 7100 00

NO LITERALS

DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000206  5a   000003 000000
000207  aa   000000 000000
000210  aa   000003 000000
000211  55   000011 000002
000212  5a   000002 400003
000213  55   000006 000010
000214  aa   006 142 145 154
000215  aa   154 145 162 000
```

```
000216  55   000015 000035
000217  aa   000204 500000
000220  55   000013 000003
000221  aa   142 142 145 164          better
000222  aa   164 145 162 000
000223  55   000002 000010
000224  aa   000000 400002
000225  55   000020 000063
000226  aa   114 163 171 155        . symbol_table
000227  aa   142 157 154 137
000230  aa   164 141 142 154
000231  aa   145 000 000 000

NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

000232  aa   000001 000000
000233  aa   000000 000000

INTERNAL EXPRESSION WORDS

@014LINKAGE INFORMATION

000000  aa   000000 000000
000001  0a   000206 000000
000002  aa   000000 000000
000003  aa   000000 000000
000004  aa   000000 000000
000005  aa   000000 000000
000006  22   000010 000010
000007  a2   000000 000010
@014SYMBOL INFORMATION

SYMBOL TABLE HEADER

000000  aa   000000 000001
000001  aa   163171 155142
000002  aa   164162 145145
000003  aa   000000 000004
000004  aa   000000 100422
000005  aa   031241 730745
000006  aa   000000 100434
000007  aa   206142 401614
000010  aa   141154 155040
000011  aa   040040 040040
000012  aa   000024 000040
000013  aa   000034 000040
000014  aa   000044 000100
000015  aa   000002 000002
000016  aa   000064 000000
000017  aa   000000 000130
000020  aa   000000 000102
000021  aa   000000 000117
000022  aa   000122 000102
000023  aa   000064 000000
000024  aa   101114 115040
000025  aa   126145 162163
000026  aa   151157 156040
000027  aa   064056 061054
000030  aa   040116 157166
000031  aa   145155 142145
000032  aa   162040 061071
000033  aa   067062 040040
```

```
000034  aa   115141 162147
000035  aa   145162 050104
000036  aa   162165 151144
000037  aa   056141 040040
000040  aa   040040 040040
000041  aa   040040 040040
000042  aa   040040 040040
000043  aa   040040 040040
000044  aa   154151 163164
000045  aa   040040 163171
000046  aa   155142 157154
000047  aa   163040 040156
000050  aa   145167 137143
000051  aa   141154 154040
000052  aa   040156 145167
000053  aa   137157 142152
000054  aa   145143 164040
000055  aa   040040 040040
000056  aa   040040 040040
000057  aa   040040 040040
000060  aa   040040 040040
000061  aa   040040 040040
000062  aa   040040 040040
000063  aa   040040 040040
000064  aa   000000 000001
000065  aa   000000 000001
000066  aa   000072 000037
000067  aa   004341 171504
000070  aa   000000 100434
000071  aa   206151 000000
000072  aa   076140 144144          >pdd>IBDqGFzfGGBBBBB>better.alm
000073  aa   076041 102104
000074  aa   161102 105172
000075  aa   144132 102102
000076  aa   102102 102102
000077  aa   076142 145164
000100  aa   164145 162056
000101  aa   141154 155040
```



```
0314              MULTICS ASSEMBLY CROSS REFERENCE LISTING

   Value        Symbol                 Source file   Line number

       0        better                 better:        2,    8.
      12        big_loop               better:       19,   40,   59.
      22        bl_entry               better:       17,   22,   28.
     150        compare_table          better:       83,  134.
      54        dividend               better:        5,   50,   71,   77,   91,  105.
     135        error_return           better:       24,  116.
     142        food                   better:       33,   35,  120.
     143        671b25                 better:       34,  125.
      60        linkptr                better:        7,   15,   64.
      65        loop                   better:       68,   84,   89,   96.
      50        lost_bit               better:        3,   16,   23,   26,   49.
     172        mask_table             better:       53,  154.
      55        mid_loop               better:       57,   63,   70.
      63        mid_loop_entry         better:       56,   61.
     144        minus_one              better:      117,  127.
     146        modulus                better:       21,  131.
      56        olly                   better:        6,   20,   29,   47.
     134        return                 better:      112,  119.
      53        set_ones               better:       44,   46.
     161        set_ones_table         better:       48,   54,  144.
      52        someplace              better:        4,   32,   37,   41.
```

APPENDIX D
Improved Password Scrambling Listing and Documentation

This appendix contains the listing of "encipher_", the improved password scrambling algorithm which was implemented on Multics following the ESD security analysis. This program is also used to encrypt and decrypt files in Multics using the standard Multics commands "encode" and "decode".

The algorithm generates a new key word by forming a function selection word from the last ciphertext word (or initial key at the start), then using the last ciphertext word as a fill, generates a new key word according to bits 0-4 of the function selection word as shown in the table below. The notation used in the table is:

- Φ rotate function (circular shift the value on the right by the amount on the left)
- $+$ addition
- $\oplus$ exclusive OR

The expressions are evaluated from right to left with parenthetic grouping having its normal meaning. As an example, the expression $M5 + A5 \Phi (M4 \oplus M3 + A3 \Phi (M2 \oplus M1 + A1 \Phi C))$ would be evaluated as

a) Rotate C by the amount A1
b) Add M1
c) Exclusive OR M2
d) Rotate the value obtained thus far by the amount A3
e) Add M3
f) Exclusive OR M4
g) Rotate the value obtained thus far by the amount A5
h) Add M5

The values of M1, ..., M7 and A1, ..., A7 are offsets in the register containing the key. The contents of this register are obtained by applying a Tausworth pseudo-random number generator (1) to the input key value. The value of C is the word that is to be enciphered.

Figure 4 is a flowchart for the portion of encipher_ that actually performs the enciphering of a word.

---

(1) Whittleself, John R.B., "A Comparison of the Correlation Behavior of Random Number Generators for the IBM 360", Communications of the ACM, Vol 11, No. 9, September 1968.

Function Select = 0-4 of $M7 \odot A7 \oplus (M6 + A6 \oplus C(i-1))$

BITS 0-4 OF FUNCTION SELECT (bits numbered 4, 3, 2, 1)	KEY GENERATING FUNCTION
0000	$M5 + A5 \oplus (M4 \odot A4 \oplus (M3 + A3 \oplus (M2 \odot A2 \oplus (M1 + A1 \oplus C))))$
0001	$M5 + M4 \odot A4 \oplus (M3 + A3 \oplus (M2 \odot A2 \oplus (M1 + A1 \oplus C)))$
0010	$M5 + A5 \oplus (M4 \odot M3 + A3 \oplus (M2 \odot A2 \oplus (M1 + A1 \oplus C)))$
0011	$M5 + M4 \odot M3 + A3 \oplus (M2 \odot A2 \oplus (M1 + A1 \oplus C))$
0100	$M5 + A5 \oplus (M4 \odot A4 \oplus (M3 + M2 \odot A2 \oplus (M1 + A1 \oplus C)))$
0101	$M5 + M4 \odot A4 \oplus (M3 + M2 \odot A2 \oplus (M1 + A1 \oplus C))$
0110	$M5 + A5 \oplus (M4 \odot M3 + M2 \odot A2 \oplus (M1 + A1 \oplus C))$
0111	$M5 + M4 \odot M3 + M2 \odot A2 \oplus (M1 + A1 \oplus C)$
1000	$M5 + A5 \oplus (M4 \odot A4 \oplus (M3 + A3 \oplus (M2 \odot M1 + A1 \oplus C)))$
1001	$M5 + M4 \odot A4 \oplus (M3 + A3 \oplus (M2 \odot M1 + A1 \oplus C))$
1010	$M5 + A5 \oplus (M4 \odot M3 + A3 \oplus (M2 \odot M1 + A1 \oplus C))$
1011	$M5 + M4 \odot M3 + A3 \oplus (M2 \odot M1 + A1 \oplus C)$
1100	$M5 + A5 \oplus (M4 \odot A4 \oplus (M3 + M2 \odot M1 + A1 \oplus C))$
1101	$M5 + M4 \odot A4 \oplus (M3 + M2 \odot M1 + A1 \oplus C)$
1110	$M5 + A5 \oplus (M4 \odot M3 + M2 \odot M1 + A1 \oplus C)$
1111	$M5 + M4 \odot M3 + M2 \odot M1 + A1 \oplus C$

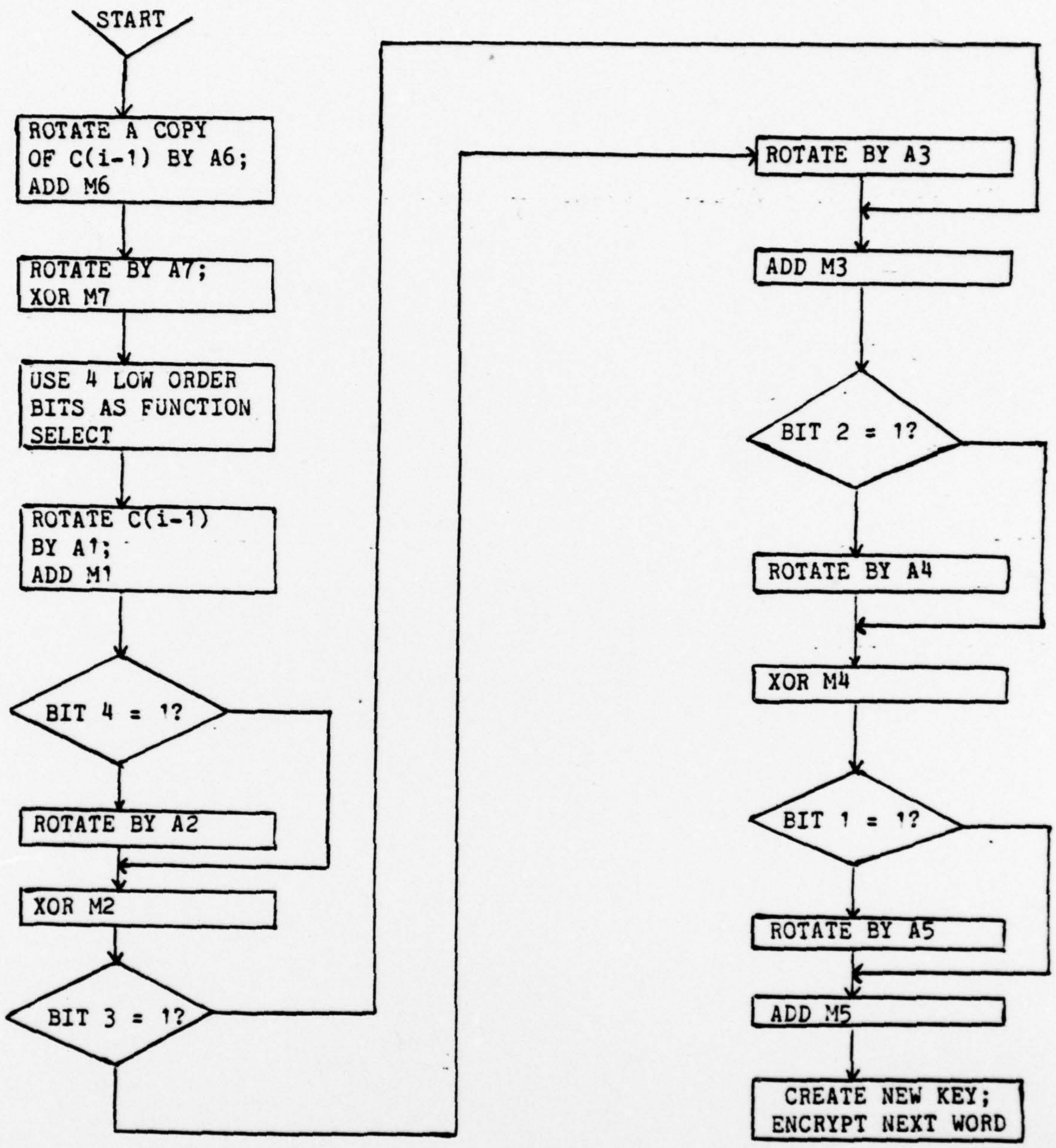


FIGURE 4. Flowchart for encipher_.

```
ASSEMBLY LISTING OF SEGMENT >udd>SDruid>Austell>encipher_.alm
ASSEMBLED ON:         05/09/75  1642.2 edt Fri
OPTIONS USED:         ls  symbols  new_call  new_object
ASSEMBLED BY:         ALM Version 4.5, September 1974
ASSEMBLER CREATED:    04/29/75  1343.9 edt Tue

                                 "****************************************************************************
                                 " This procedure enciphers an arrya of double words, i.e., fixed bin(71),
                                 " using the key that is provided.  It has entries to both encipher and decipher:
                                 "
                                 "          call encipher_(key,input_array,output_array,array_length)
                                 "
                                 "          call decipher_(key,input_array,output_array,array_length)
                                 "
                                 " where:   key                      is fixed bin(71) key for coding
                                 "          input_array(array_length) is fixed bin(71) array
                                 "          output_array(array_length) is fixed bin(71) array
                                 "          array_length             is fixed bin(17) length (double words) of array
                                 "
                                 "          Coded 1 April 1973 by Roger R. Schell, Major, USAF
                                 "****************************************************************************

000000                                      followon
000000                 000000               entry     encipher_
000000                 000004               entry     decipher_

000000                 000002               equ       key,2
000000                 000004               equ       input_array,4
000000                 000006               equ       output_array,6
000000                 000010               equ       array_length,8

                                 "
                                 "          Entry to encipher
                                 "

000000                           encipher_:
000000  aa    000100 6270 00                push
000001  aa  7 00040 2721 20
000002  aa  0 00006 3701 20                 epplp     ap|output_array,*     "LP -> cipher text
000003  0a    000007 7100 00                tra       setup_keys

                                 "
                                 "          Entry to decipher
                                 "

000004                           decipher_:
000004  aa    000100 6270 00                push
000005  aa  7 00040 2721 20
000006  aa  0 00004 3701 20                 epplp     ap|input_array,*      "set LP -> cipher text

000007                           setup_keys:
                                                                      "First create internal keying variables
                                 "Use Tausworth pseudo-random number generator on key

000007                 000013               equ       shift,11          "Shift for generator
000007                 000044               equ       size,36           "Word size used for generator

                                            tempd     variables(12)     "Internal keying variables

000007  aa    000000 6260 00                eax6      0                 "loop index in x6
```

```
000010  aa  0 00002 2371 20            ldaq    aplkey,*            "Start with input key

000011                      mask_loop:
                                                                   "Create masks
000011  aa  6 00050 7571 16            staq    variables,6         "save copy of generator seed
000012  aa    000013 7720 00           qrl     shift               "Now generate pseudo-random number
000013  aa    000013 7710 00           arl     shift
000014  aa  6 00050 6771 16            eraq    variables,6
000015  aa  6 00050 7571 16            staq    variables,6
000016  aa    000031 7360 00           qls     size-shift
000017  aa    000031 7350 00           als     size-shift
000020  aa  6 00050 6771 16            eraq    variables,6
000021  aa  6 00050 7571 16            staq    variables,6         "Save result

000022  aa    000002 6260 16           eax6    2,6
000023  aa    000022 1060 03           cmpx6   18,du               "Generate 9 double words
000024  0a    000011 6010 00           tnz     mask_loop

                            "
                            "Next create 7-bit shift variables

000025  aa    000000 6260 00           eax6    0
000026  aa    000013 7730 00           lrl     11                  "First 7 bits to upper A-reg
000027  aa    000000 6200 00           eax0    0                   "Zero for clearing half word
000030                      shift_loop:
000030  aa  6 00070 7551 16            sta     variables+A1,6      "Upper A-reg is shift variable
000031  aa  6 00070 4401 16            sxl0    variables+A1,6      "Zero lower half word
000032  aa    000007 7370 00           lls     7
000033  0a    000116 3750 00           ana     =o000177777777      "Save 7 bits in upper A-reg
000034  aa    000001 6260 16           eax6    1,6
000035  aa    000007 1060 03           cmpx6   7,du                "Generate 7 shift variatles
000036  0a    000030 6010 00           tnz     shift_loop

                            "
                            "        Now that we have needed variables, aply the cipher
                            "

                            "Declaration of offsets of keying variables
000037                000000           equ     C0,0                "Initial cipher text from key
000037                000002           equ     M1,2                "Mask variables
000037                000004           equ     M2,4
000037                000006           equ     M3,6
000037                000010           equ     M4,8
000037                000012           equ     M5,10
000037                000014           equ     M6,12
000037                000016           equ     M7,14
000037                000020           equ     A1,16               "Amount of shift -- as address
000037                000021           equ     A2,17
000037                000022           equ     A3,18
000037                000023           equ     A4,19
000037                000024           equ     A5,20
000037                000025           equ     A6,21
000037                000026           equ     A7,22

000037  aa  0 00010 7251 20            lxl5    aplarray_length,*   "Get length (double words)
000040  aa    777777 6250 15           eax5    -1,5                "Check for zero or negative
000041  0a    000103 6040 00           tmi     return
000042  aa    000000 6260 00           eax6    0                   "X6 is index into arrays
000043  aa  6 00050 3521 00            eppbp   variables+C0        "Initial cipher text from key
000044                      cipher_loop:
```

```
000044  aa  2 00000 2371 00              ldaq      bpl0

                                  "First compute select function

000045  aa  6 00075 7771 20              llr       variables+A6,*
000046  aa  6 00064 0371 00              adlaq     variables+M6
000047  aa  6 00076 7771 20              llr       variables+A7,*
000050  aa  6 00066 6771 00              eraq      variables+M7
000051  aa    000000 6210 06             eax1      0,ql                   "Save select function
                                  "
                                  "Compute value
                                  "
000052  aa  2 00000 2371 00              ldaq      bpl0
000053  aa  6 00070 7771 20              llr       variables+A1,*
000054  aa  6 00052 0371 00              adlaq     variables+M1
000055  aa    000010 3010 03             canx1     =o10,du
000056  aa    000002 6010 04             tnz       2,ic
000057  aa  6 00071 7771 20              llr       variables+A2,*
000060  aa  6 00054 6771 00              eraq      variables+M2
000061  aa    000004 3010 03             canx1     =o4,du
000062  aa    000002 6010 04             tnz       2,ic
000063  aa  6 00072 7771 20              llr       variables+A3,*
000064  aa  6 00056 0371 00              adlaq     variables+M3
000065  aa    000002 3010 03             canx1     =o2,du
000066  aa    000002 6010 04             tnz       2,ic
000067  aa  6 00073 7771 20              llr       variables+A4,*
000070  aa  6 00060 6771 00              eraq      variables+M4
000071  aa    000001 3010 03             canx1     =o1,du
000072  aa    000002 6010 04             tnz       2,ic
000073  aa  6 00074 7771 20              llr       variables+A5,*
000074  aa  6 00062 0371 00              adlaq     variables+M5           "AQ contains computed key

000075  aa  4 00000 3521 16              eppbp     lpl0,6                 "set BP -> next cipher text autokey
000076  aa  0 00004 6771 76              eraq      ap|input_array,*6
000077  aa  0 00006 7571 76              staq      ap|output_array,*6     "return ciphered value
000100  aa    000002 6260 16             eax6      2,6                    "Increment array offset
000101  aa    777777 6250 15             eax5      -1,5                   "Check for end of array
000102  0a    000044 6050 00             tpl       cipher_loop
000103                            return:
                                  "
                                  "Clean up the 'dirty blackboard' before returning

000103                005202             bool      rpt,5202               "RPT instruction

000103  0a    000103 2370 00             ldaq      *                      " Load AQ with garbage
000104  aa    000000 6260 00             eax6      0
000105  aa    026200 520202              vfd       8/11,2/0,1/1,7/0,12/rpt,6/2 "RPT instruction
000106  aa  6 00050 7571 16              staq      variables,6            "Overwrite keying variables

000107  aa  7 00042 7101 20              return

                                         end
```

ENTRY SEQUENCES

000110	5a	000017	0000	00
000111	aa	7 00046	2721	20
000112	0a	000000	7100	00
000113	5a	000011	0000	00
000114	aa	7 00046	2721	20
000115	0a	000004	7100	00

LITERALS

```
000116  aa   000177 777777
```

```
NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

     000117  5a    000003 000000
     000120  aa    000000 600000
     000121  aa    000000 000000
     000122  55    000011 000002
     000123  5a    000002 400003
     000124  55    000006 000011
     000125  aa   011 145 156 143
     000126  aa   151 160 150 145
     000127  aa   162 137 000 000
     000130  55    000017 000003
     000131  0a    000114 500000
     000132  55    000014 000003
     000133  aa   011 144 145 143          decipher_
     000134  aa   151 160 150 145
     000135  aa   162 137 000 000
     000136  55    000025 000011
     000137  0a    000111 500000
     000140  55    000022 000003
     000141  aa   011 145 156 143          encipher_
     000142  aa   151 160 150 145
     000143  aa   162 137 000 000
     000144  55    000002 000017
     000145  6a    000000 400002
     000146  55    000030 000003
     000147  aa   014 163 171 155          symbol_table
     000150  aa   142 157 154 137
     000151  aa   164 141 142 154
     000152  aa   145 000 000 000

NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

     000153  aa    000001 000000
     000154  aa    000000 000000

INTERNAL EXPRESSION WORDS

     000155  aa    000000 000000
```

LINKAGE INFORMATION

```
000000  aa  000000 000000
000001  0a  000117 000000
000002  aa  000000 000000
000003  aa  000000 000000
000004  aa  000000 000000
000005  aa  000000 000000
000006  22  000010 000010
000007  a2  000000 000010
```

SYMBOL INFORMATION

SYMBOL TABLE HEADER

```
000 000  aa   000000 000001
000 001  aa   163171 155142
000 002  aa   164162 145145
000 003  aa   000000 000004
000 004  aa   000000 102523
000 005  aa   146512 715066
000 006  aa   000000 102537
000 007  aa   733521 472051
000 010  aa   141154 155040
000 011  aa   040040 040040
000 012  aa   000024 000040
000 013  aa   000034 000040
000 014  aa   000044 000100
000 015  aa   000002 000002
000 016  aa   000064 000000
000 017  aa   000000 000124
000 020  aa   000000 000103
000 021  aa   000000 000113
000 022  aa   000116 000103
000 023  aa   000064 000600
000 024  aa   101114 115040
000 025  aa   126145 162163
000 026  aa   151157 156040
000 027  aa   064056 065054
000 030  aa   040123 145160
000 031  aa   164145 155142
000 032  aa   145162 040061
000 033  aa   071067 064040
000 034  aa   101165 163164
000 035  aa   145154 154056
000 036  aa   123104 162165
000 037  aa   151144 056141
000 040  aa   040040 040040
000 041  aa   040040 040040
000 042  aa   040040 040040
000 043  aa   040040 040040
000 044  aa   154163 040040
000 045  aa   163171 155142
000 046  aa   157154 163040
000 047  aa   040156 145167
000 050  aa   137143 141154
000 051  aa   154040 040156
000 052  aa   145167 137157
000 053  aa   142152 145143
000 054  aa   164040 040040
000 055  aa   040040 040040
000 056  aa   040040 040040
000 057  aa   040040 040040
000 060  aa   040040 040040
000 061  aa   040040 040040
000 062  aa   040040 040040
000 063  aa   040040 040040
000 064  aa   000000 000001
000 065  aa   000000 000001
000 066  aa   000072 000041
000 067  aa   025376 657514
```

```
000070  aa   000000 102537
000071  aa   733524 600000
000072  aa   076165 144144
000073  aa   076123 104162
000074  aa   165151 144076
000075  aa   101165 163164
000076  aa   145154 154076
000077  aa   145156 143151
000100  aa   160150 145162
000101  aa   137056 141154
000102  aa   155040 040040
```

>udd>SOruld>Austell>encipher_.alm

MULTICS ASSEMBLY CROSS REFERENCE LISTING

```
Value     Symbol              Source file   Line number

   20     A1                  encipher_!     78,  79,  99, 126.
   21     A2                  encipher_!    100, 130.
   22     A3                  encipher_!    101, 134.
   23     A4                  encipher_!    102, 138.
   24     A5                  encipher_!    103, 142.
   25     A6                  encipher_!    104, 117.
   26     A7                  encipher_!    105, 119.
   10     array_length        encipher_!     24, 107.
    0     C0                  encipher_!     91, 111.
   44     cipher_loop         encipher_!    112, 150.
    4     decipher_           encipher_!     19,  39.
    0     encipher_           encipher_!     18,  30.
    4     input_array         encipher_!     22,  41, 146.
    2     key                 encipher_!     21,  53.
    2     M1                  encipher_!     92, 127.
    4     M2                  encipher_!     93, 131.
    6     M3                  encipher_!     94, 135.
   10     M4                  encipher_!     95, 139.
   12     M5                  encipher_!     96, 143.
   14     M6                  encipher_!     97, 118.
   16     M7                  encipher_!     98, 120.
   11     mask_loop           encipher_!     55,  69.
    6     output_array        encipher_!     23,  32, 147.
  103     return              encipher_!    109, 151.
 5202     rpt                 encipher_!    155, 159.
    7     setup_keys          encipher_!     33,  43.
   13     shift               encipher_!     47,  58,  59,  62,  63.
   30     shift_loop          encipher_!     77,  84.
   44     size                encipher_!     48,  62,  63.
   50     variables           encipher_!     50,  57,  60,  61,  64,  65,  78,  79, 111, 117, 118, 119,
                                            120, 126, 127, 130, 131, 134, 135, 138, 139, 142, 143, 160.
```

NO FATAL ERRORS

MISSION

OF THE

DIRECTORATE OF COMPUTER SYSTEMS ENGINEERING

The Directorate of Computer Systems Engineering provides ESD with technical services on matters involving computer technology to help ESD system development and acquisition offices exploit computer technology through engineering application to enhance Air Force systems and to develop guidance to minimize R&D and investment costs in the application of computer technology.

The Directorate of Computer Systems Engineering also supports AFSC to insure the transfer of computer technology and information throughout the Command, including maintaining an overview of all matters pertaining to the development, acquisition, and use of computer resources in systems in all Divisions, Centers and Laboratories and providing AFSC with a corporate memory for all problems/solutions and developing recommendations for RDT&E programs and changes in management policies to insure such problems do not reoccur.