

```

LLL                                0000000000    GGGGGGGGGGGG    IIIIIIIII    NNN    NNN
LLL                                0000000000    GGGGGGGGGGGG    IIIIIIIII    NNN    NNN
LLL                                0000000000    GGGGGGGGGGGG    IIIIIIIII    NNN    NNN
LLL                                000          000    GGG          III          NNN    NNN
LLL                                000          000    GGG          III          NNN    NNN
LLL                                000          000    GGG          III          NNN    NNN
LLL                                000          000    GGG          III          NNNNNN    NNN
LLL                                000          000    GGG          III          NNNNNN    NNN
LLL                                000          000    GGG          III          NNNNNN    NNN
LLL                                000          000    GGG          III          NNN    NNN    NNN
LLL                                000          000    GGG          III          NNN    NNN    NNN
LLL                                000          000    GGG          III          NNN    NNN    NNN
LLL                                000          000    GGG          III          NNN    NNNNNN
LLL                                000          000    GGG          III          NNN    NNNNNN
LLL                                000          000    GGG          III          NNN    NNNNNN
LLL                                000          000    GGG          III          NNN    NNN
LLL                                000          000    GGG          III          NNN    NNN
LLL                                000          000    GGG          III          NNN    NNN
LLL                                000          000    GGG          III          NNN    NNN
LLLLLLLLLLLLLLLLLLLL            0000000000    GGGGGGGGGG    IIIIIIIII    NNN    NNN
LLLLLLLLLLLLLLLLLLLL            0000000000    GGGGGGGGGG    IIIIIIIII    NNN    NNN
LLLLLLLLLLLLLLLLLLLL            0000000000    GGGGGGGGGG    IIIIIIIII    NNN    NNN

```

```
HH      HH  PPPPPPP  WW      WW  DDDDDDDD
HH      HH  PPPPPPP  WW      WW  DDDDDDDD
HH      HH  PP      PP  WW      WW  DD      DD
HH      HH  PP      PP  WW      WW  DD      DD
HH      HH  PP      PP  WW      WW  DD      DD
HH      HH  PP      PP  WW      WW  DD      DD
HHHHHHHHHH PPPPPPP  WW      WW  DD      DD
HHHHHHHHHH PPPPPPP  WW      WW  DD      DD
HH      HH  PP      WW  WW  WW  DD      DD
HH      HH  PP      WW  WW  WW  DD      DD
HH      HH  PP      WWW  WWW  DD      DD
HH      HH  PP      WWW  WWW  DD      DD
HH      HH  PP      WW      WW  DDDDDDDD
HH      HH  PP      WW      WW  DDDDDDDD
```

....  
....  
....  
....

```
LL      IIIIII  SSSSSSSS
LL      IIIIII  SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL IIIIII  SSSSSSSS
LLLLLLLLLL IIIIII  SSSSSSSS
```

(2) 49  
(3) 98  
(4) 212

DECLARATIONS  
Dispatch - select encryption algorithm  
Purdy - evaluate Purdy polynomial

```
0000 1 .TITLE HPWD - hash user password
0000 2 .IDENT 'V04-000'
0000 3
0000 4 :
0000 5 :*****
0000 6 :*
0000 7 :* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8 :* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9 :* ALL RIGHTS RESERVED.
0000 10 :*
0000 11 :* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12 :* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13 :* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14 :* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15 :* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16 :* TRANSFERRED.
0000 17 :*
0000 18 :* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19 :* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20 :* CORPORATION.
0000 21 :*
0000 22 :* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23 :* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24 :*
0000 25 :*
0000 26 :*****
0000 27 :
0000 28 :++
0000 29 : FACILITY: User Verification Subroutine
0000 30 :
0000 31 : ABSTRACT:
0000 32 :
0000 33 : ENVIRONMENT:
0000 34 :
0000 35 : AUTHOR: H. M. Levy , CREATION DATE: 20-Sep-1977
0000 36 :
0000 37 : MODIFIED BY:
0000 38 :
0000 39 : V03-002 SHZ0002 Stephen H. Zalewski, 02-Mar-1984
0000 40 : If zero length password is passed in, return a null password.
0000 41 : If PURGY_V algorithm used, add check to prevent us from searching
0000 42 : beyond length of username buffer.
0000 43 :
0000 44 : V03-001 SHZ0001 Stephen H. Zalewski 14-Feb-1984
0000 45 : Add support for PURDY and PURDY_V encryption.
0000 46 :
0000 47 :--
```

```
0000 49 .SBTTL DECLARATIONS
0000 50
0000 51 ;
0000 52 ; MACROS:
0000 53 ;
0000 54 ;
0000 55 .macro pushq Src
0000 56 movq Src,-(sp)
0000 57 .endm
0000 58 .macro popq Dst
0000 59 movq (sp)+,Dst
0000 60 .endm
0000 61
0000 62 ;
0000 63 ; EQUATED SYMBOLS:
0000 64 ;
0000 65
00000004 0000 66 OUTDSC = 4 ; adr of encrypted output descriptor
00000008 0000 67 PWDDSC = OUTDSC + 4 ; adr of password descriptor
0000000C 0000 68 ENCRYPT = PWDDSC + 4 ; encryption algorithm index (byte)
00000010 0000 69 SALT = ENCRYPT + 4 ; random number (word)
00000014 0000 70 USRDSC = SALT + 4 ; adr of username descriptor
0000 71
0000 72 ;
0000 73 ; OWN STORAGE:
0000 74 ;
0000 75
00000000 76 .psect _LIB$CODE RD,NOWRT,PIC,SHR,BYTE,EXE
0000 77
0000 78 ;
0000 79 ; AUTODIN-II polynomial table used by CRC algorithm
0000 80 ;
0000 81
0000 82 AUTODIN:
26D930AC 3B6E20C8 1DB71064 00000000 0000 83 .LONG ^000000000000,^003555610144,^007333420310,^004666230254
5005713C 4DB26158 6B6B51F4 76DC4190 0010 84 .LONG ^016667040620,^015332650764,^011554460530,^012001270474
CB61B38C D6D6A3E8 F00F9344 EDB88320 0020 85 .LONG ^035556101440,^036003711504,^032665521750,^031330331614
BDBDF21C A00AE278 86D3D2D4 9B64C2B0 0030 86 .LONG ^023331141260,^020664751324,^024002561170,^027557371034
0040 87
0040 88
0040 89 ; The following table of coefficients is used by the Purdy polynomial
0040 90 ; algorithm. They are prime, but the algorithm does not require this.
0040 91
FFFFFFF FFFFFFFAD 0040 92 C: .long -83, -1 ; C1
FFFFFFF FFFFFFF4D 0048 93 .long -179, -1 ; C2
FFFFFFF FFFFFFFE95 0050 94 .long -257, -1 ; C3
FFFFFFF FFFFFFFE95 0058 95 .long -323, -1 ; C4
FFFFFFF FFFFFFFE95 0060 96 .long -363, -1 ; C5
```

```
0068 98 .SBTTL Dispatch - select encryption algorithm
0068 99
0068 100 :++
0068 101 :
0068 102 : FUNCTIONAL DESCRIPTION:
0068 103 :
0068 104 :     Smash up the password into a non-reversible number.
0068 105 :
0068 106 : CALLINGS SEQUENCE:
0068 107 :
0068 108 :     CALLS/CALLG
0068 109 :
0068 110 : INPUTS:
0068 111 :
0068 112 :     PWDDSC - Address of password descriptor
0068 113 :     ENCRYPT - Encryption algorithm index (byte)
0068 114 :     SALT - random number (word)
0068 115 :     USRDSC - Address of username descriptor
0068 116 :
0068 117 : IMPLICIT INPUTS:
0068 118 :
0068 119 :     none
0068 120 :
0068 121 : OUTPUTS:
0068 122 :
0068 123 :     OUTDSC - Address of output buffer descriptor
0068 124 :
0068 125 : IMPLICIT OUTPUTS:
0068 126 :
0068 127 :     none
0068 128 :
0068 129 : ROUTINE VALUE:
0068 130 :
0068 131 :     Success status
0068 132 :
0068 133 : SIDE EFFECTS:
0068 134 :
0068 135 :     none
0068 136 :--
0068 137
007C 0068 138 .entry LGI$HPWD,^M<r2,r3,r4,r5,r6> ; entry mask
006A 139
006A 140     tstb ENCRYPT(ap) ; using CRC algorithm?
006D 141     beql 20$ ; yes, no processing of user desc necessary.
006F 142     subl2 #20,sp ; Get temp desc and buffer off stack
0072 143     movl sp,r6 ; Put address into r6
0075 144     movq @USRDSC(ap),(r6) ; Put current user desc into stack desc.
0079 145     cmpb #1,ENCRYPT(ap) ; Which PURDY algorithm?
007D 146     bneq 10$
007F 147     movc5 (r6),a4(r6),#32,#12,8(r6) ; PURDY. 12 character blank pad username
0087 148     movw #12,(r6) ; size of desc is now 12
008A 149     movab 8(r6),4(r6) ; Point desc to buffer on stack.
008F 150     brb 20$ ; goto main line
0091 151
0091 152
0091 153 10$: movzwl (r6),r5 ; PURDY V. Remove padding from username.
0094 154     clrw (r6) ; Save length of username in r5.
```

|    |    |    |    |    |    |      |      |       |                |
|----|----|----|----|----|----|------|------|-------|----------------|
| 0B | A6 | OC | 20 | 04 | B6 | 66   | 2C   | 007F  | 147            |
|    |    |    |    | 66 | OC | B0   | 0087 | 148   |                |
|    |    | 04 | A6 | 08 | A6 | 9E   | 008A | 149   |                |
|    |    |    |    |    | 1C | 11   | 008F | 150   |                |
|    |    |    |    |    |    |      | 0091 | 151   |                |
|    |    |    |    |    |    |      | 0091 | 152   |                |
|    |    | 55 |    | 66 | 3C | 0091 | 153  | 10\$: | movzwl (r6),r5 |
|    |    |    |    | 66 | B4 | 0094 | 154  |       | clrw (r6)      |

```

50 04 A6 D0 0096 155      movl    4(r6),r0      ; Get address of username buffer into R0.
20 80 91 009A 156 15$:    cmpb    (r0)+,#32      ; Search until we find first blank.
OE 13 009D 157          beql     20$
66 66 B6 009F 158          incw    (r6)          ; Increment byte count until blank found
1F B1 00A1 159          cmpw     #31,(r6)        ; or 31 characters have been encountered
07 13 00A4 160          beql     20$            ; (31 is max username length).
66 55 B1 00A6 161          cmpw     r5,(r6)        ; or entire buffer has been parsed.
02 13 00A9 162          beql     20$            ; (someone not playing by the rules)
ED 11 00AB 163          brb      15$
      00AD 164
54 08 BC 7E 00AD 165 20$: movaq    @PWDDSC(ap),r4    ; If password is zero length
64 D5 00B1 166          tstl     (r4)
OF 12 00B3 167          bneq     25$
54 04 BC 7E 00B5 168          movaq    @OUTDSC(ap),r4    ; Then return null password
64 B4 00B9 169          clrw     (r4)
04 B4 08 00 64 00 2C 00BB 170          movc5    #0,(r4),#0,#8,@4(r4) ; (quadword of zeros)
3E 11 00C2 171          brb      40$
      00C4 172
54 04 BC 7E 00C4 173 25$: movaq    @OUTDSC(ap),r4    ; Get pointer to output buffer
54 04 B4 7E 00C8 174          movaq    @4(r4),r4
OC AC 95 00CC 175          tstb     ENCRYPT(ap)      ; Use the CRC algorithm if the index
16 1A 00CF 176          bgtru     30$              ; is zero
50 01 CE 00D1 177          mnegl     #1,r0          ; initial CRC
51 08 BC 7E 00D4 178          movaq    @PWDDSC(ap),r1    ; get descriptor address
04 B1 61 50 FF24 CF 0B 00D8 179          crc      AUTODIN,r0,(r1),@4(r1) ; convert password to 32-bit number
51 D4 00E0 180          clrl     r1                ; high order longword will be zero
64 50 7D 00E2 181          movq     r0,(r4)          ; copy result to the output buffer
1B 11 00E5 182          brb      40$
      00E7 183
      64 7C 00E7 184 30$: clrq     (r4)              ; Initialize output buffer
53 08 BC 7E 00E9 185          movaq    @PWDDSC(ap),r3    ; Collapse password to a quadword
17 10 00ED 186          bsbb      COLLAPSE_R2
03 A4 10 AC A0 00EF 187          addw2    SALT(ap),3(r4)    ; Add random salt into middle of U
53 56 D0 00F4 188          movl     r6,r3            ; Collapse username into the quadword
OD 10 00F7 189          bsbb      COLLAPSE_R2
64 7F 00F9 190          pushaq   (r4)              ; Push pointer to U
0000011F'EF 01 FB 00FB 191          calls   #1,Purdy      ; Run U through the polynomial mod P
50 01 D0 0102 192          0102 192
04 0105 193 40$:      movl     #1,r0
      0106 194          ret
      0106 195
      0106 196 COLLAPSE_R2:
      0106 197 .enabl  [LSB]
      0106 198 ; This routine takes a string of bytes (the descriptor for which is pointed
      0106 199 ; to by r3) and collapses them into a quadword (pointed to by r4). It does
      0106 200 ; this by cycling around the bytes of the output buffer adding in the bytes
      0106 201 ; of the input string.
      0106 202
50 63 3C 0106 203          movzwl   (r3),r0          ; Obtain the number of input bytes
13 13 0109 204          beqlu     20$
52 04 B3 DE 010B 205          moval    @4(r3),r2          ; Obtain pointer to input string
51 50 FFFFFFFF8 8F CB 010F 206 10$: bicl3    #-8,r0,r1    ; Obtain cyclic index into output buffer
6441 82 80 0117 207          addb2    (r2)+,(r4)[r1]
F1 50 F5 011B 208          sobgtr   r0,10$          ; Loop until input string is exhausted
05 011E 209 20$:      rsb
      011F 210 .dsabl  LSB
```

```
011F 212 .sbtll Purdy - evaluate Purdy polynomial
011F 213
0000003B 011F 214 a=59 ; 2^64 - 59 is the biggest quadword prime
011F 215
00FFFFFFD 011F 216 n0=1a24 - 3 ; These exponents are prime, but this is
00FFFFFFC1 011F 217 n1=1a24 - 63 ; not required by the algorithm.
011F 218
003C 011F 219 .entry Purdy,^M<r2,r3,r4,r5>
0121 220 ;
0121 221 ; This routine computes f(U) = p(U) mod P. Where P is a prime of the form
0121 222 ; P = 2^64 - a. The function p is the following polynomial:
0121 223 ; X^n0 + X^n1*C1 + X^3*C2 + X^2*C3 + X*C4 + C5
0121 224 ; The input U is an unsigned quadword.
0121 225 ;
0121 226 ;
0121 227 pushq @4(ap) ; Push U
0125 228 bsbw PQMOD_R0 ; Ensure U less than P
0128 229 movaq (sp),r4 ; Maintain a pointer to X
012B 230 movaq C,r5 ; Point to the table of coefficients
0130 231 pushq (r4)
0133 232 pushl #n1 ; X^n1
0139 233 bsbw PQEXP_R3
013B 234 pushq (r4)
013E 235 pushl #n0-n1
0140 236 bsbw PQEXP_R3
0142 237 pushq (r5)+ ; C1
0145 238 bsbw PQADD_R0 ; X^(n0 - n1) + C1
0148 239 bsbw PQMUL_R2 ; X^n0 + X^n1*C1
014B 240 pushq (r5)+ ; C2
014E 241 pushq (r4)
0151 242 bsbw PQMUL_R2 ; X*C2
0154 243 pushq (r5)+ ; C3
0157 244 bsbw PQADD_R0 ; X*C2 + C3
015A 245 pushq (r4)
015D 246 bsbw PQMUL_R2 ; X^2*C2 + X*C3
015F 247 pushq (r5)+ ; C4
0162 248 bsbw PQADD_R0 ; X^2*C2 + X*C3 + C4
0165 249 pushq (r4)
0168 250 bsbw PQMUL_R2 ; X^3*C2 + X^2*C3 + C4*X
016A 251 pushq (r5)+ ; C5
016D 252 bsbw PQADD_R0 ; X^3*C2 + X^2*C3 + C4*X + C5
0170 253 bsbw PQADD_R0 ; Add in the high order terms
0173 254 popq @4(ap) ; Replace U with f(X)
50 01 D0 0177 255 movl #1,r0
04 017A 256 ret
017B 257
017B 258
017B 259 PQEXP_R3:
017B 260 .enabl LSB
017B 261 ; Replaces the inputs with U^n mod P where P is of the form P = 2^64 - a.
017B 262 ; U is a quadword, n is an unsigned longword.
017B 263
08 BA 017B 264 popr #^M<r3> ; Record return address
017D 265 pushq #1 ; Initialize
0180 266 pushq 8+4(sp) ; Copy U to top of stack for speed
10 AE D5 0184 267 tstl 8+8(sp) ; Only handle n greater than 0
27 13 0187 268 beqlu 30$
```

```
15 10 AE E9 0189 269 10$: blbc 8+8(sp),20$
                                018D 270 pushq (sp) ; Copy the current power of U
                                0190 271 pushq 8+8(sp) ; Multiply with current value
                                43 10 0194 272 bsbb PQMUL_R2
                                0196 273 popq 8(sp) ; Replace current value
00 10 AE 1F 01 ED 019A 274 cmpzv #1,#31,8+8(sp),#0
                                OE 13 01A0 275 beqlu 30$
                                01A2 276 20$: pushq (sp) ; Proceed to next power of U
10 AE 10 AE 1F 01 EF 01A5 277 bsbb PQMUL_R2
                                D9 11 01A7 278 extzv #1,#31,8+8(sp),8+8(sp)
                                14 AE 08 AE 7D 01AE 279 brb 10$
                                SE 14 AE 7E 01B0 280 30$: movq 8(sp),8+8+4(sp) ; Copy the return value
                                63 17 01B5 281 movaq 8+8+4(sp),sp ; Discard the exponent
                                01B9 282 jmp (r3) ; return
                                01BB 283 .dsabl LSB
                                01BB 284
00000000 01BB 285 u=0 ; Low longword of U
00000004 01BB 286 v=u+4 ; High longword of U
00000008 01BB 287 y=u+8 ; Low longword of Y
0000000C 01BB 288 z=y+4 ; High longword of Y
                                01BB 289
                                01BB 290 PQMOD_R0:
                                01BB 291 .enabl LSB
                                01BB 292 ; Replaces the quadword U on the stack with U mod P where P is of the
                                01BB 293 ; form P = 2^64 - a.
                                01BB 294
FFFFFFFF 8F 04 AE D1 01BB 295 popr #^M<r0> ; Record return address
                                10 1F 01BD 296 cmpl v(sp),#-1 ; Replace U with U mod P
FFFFFFFFC5 8F 06E D1 01C5 297 blssu 10$
                                07 1F 01C7 298 cmpl u(sp),#-a
                                6E 3B C0 01CE 299 blssu 10$
                                04 AE 00 D8 01D0 300 addl2 #a,u(sp)
                                60 17 01D3 301 adwc #0,v(sp)
                                01D7 302 10$: jmp (r0) ; return
                                01D9 303 .dsabl LSB
                                01D9 304
                                01D9 305
                                01D9 306 PQMUL_R2:
                                01D9 307 ; Computes the product U*Y mod P where P is of the form P = 2^64 - a.
                                01D9 308 ; U, Y are quadwords less than P. The product replaces U and Y on the stack.
                                01D9 309
                                01D9 310 ; The product may be formed as the sum of four longword multiplications
                                01D9 311 ; which are scaled by powers of 2^32 by evaluating:
                                01D9 312 ; 2^64*v*z + 2^32*(v*y + u*z) + u*y
                                01D9 313 ; The result is computed such that division by the modulus P is avoided.
                                01D9 314
52 02 BA 01D9 315 popr #^M<r1> ; Record return address
0C 5E D0 01DB 316 movl sp,r2 ; Record initial stack value
04 A2 DD 01DE 317 pushl z(r2)
04 A2 DD 01E1 318 pushl v(r2)
03 10 01E4 319 bsbb EMULQ
03 10 01E6 320 bsbb PQMOD_R0
03 10 01E8 321 bsbb PQLSH_R0 ; Obtain 2^32*v*z
08 A2 DD 01EA 322 pushl y(r2)
04 A2 DD 01ED 323 pushl v(r2)
02 10 01F0 324 bsbb EMULQ
07 10 01F2 325 bsbb PQMOD_R0
```

```
0C A2 DD 01F4 326      pushl  z(r2)
    62 DD 01F7 327      pushl  u(r2)
    1D 10 01F9 328      bsbb    EMULQ
    BE 10 01FB 329      bsbb    PQMOD_R0
    4E 10 01FD 330      bsbb    PQADD_R0      ; Obtain (v*y + u*z)
    4C 10 01FF 331      bsbb    PQADD_R0      ; Add in 2^32*v*z
    39 10 0201 332      bsbb    PQLSH_R0      ; Obtain the first two terms
08 A2 DD 0203 333      pushl  y(r2)
    62 DD 0206 334      pushl  u(r2)
    OE 10 0208 335      bsbb    EMULQ
    AF 10 020A 336      bsbb    PQMOD_R0      ; Obtain the third term: u*y
    3F 10 020C 337      bsbb    PQADD_R0      ; Add it in
    020E 338      popq     Y(r2)      ; Copy the return value
5E 08 A2 7E 0212 339      movaq  Y(r2),sp    ; Point the stack to the return value
    61 17 0216 340      jmp     (r1)      ; return
    0218 341
    0218 342
    0218 343 EMULQ:
    0218 344 .enabl  LSB
    0218 345 ; This routine knows how to multiply two unsigned longwords, replacing them
    0218 346 ; with the unsigned quadword product on the stack.
    0218 347
7E 00 08 AE 04 AE 7A 0218 348      emul    4(sp),8(sp),#0,-(sp)
    7E D4 021F 349      clrl    -(sp)
    10 AE D5 0221 350      tstl    4+8+4(sp)      ; Check both longwords to see if we must
    04 18 0224 351      bgeq    10$      ; compensate for the unsigned bias.
    6E 14 AE C0 0226 352      addl    4+8+8(sp),(sp)
    14 AE D5 022A 353 10$:  tstl    4+8+8(sp)
    04 18 022D 354      bgeq    20$
    6E 10 AE C0 022F 355      addl    4+8+4(sp),(sp)
    04 AE 8E C0 0233 356 20$:  addl    (sp)+,4(sp)      ; Add in the compensation.
    05 0237 357      popq     4(sp)      ; Replace the longwords with their product.
    023B 358      rsb
    023C 359 .dsabl  LSB
    023C 360
    023C 361
    023C 362 PQLSH_R0:
    023C 363 .enabl  LSB
    023C 364 ; Computes the product 2^32*U mod P where P is of the form P = 2^64 - a.
    023C 365 ; U is a quadword less than P. The product replaces U on the stack.
    023C 366
    023C 367 ; This routine is used by PQMUL in the formation of quadword products in
    023C 368 ; such a way as to avoid division by the modulus P.
    023C 369 ; The product 2^64*v + 2^32*u is congruent a*v + 2^32*u mod P (where u, v
    023C 370 ; are longwords).
    023C 371
    01 BA 023C 372      popr     #^M<r0>      ; Record return address
    04 AE DD 023E 373      pushl  v(sp)
    3B DD 0241 374      pushl  #a
    D3 10 0243 375      bsbb    EMULQ      ; Push a*v
    08 AE 08 AE 20 79 0245 376      ashq   #32,Y(sp),Y(sp) ; Form Y = 2^32*u
    02 11 024B 377      brb     10$      ; Return the sum U + Y mod P.
    024D 378
    024D 379 PQADD_R0:
    024D 380 ; Computes the sum U + Y mod P where P is of the form P = 2^64 - a.
    024D 381 ; U, Y are quadwords less than P. The sum replaces U and Y on the stack.
    024D 382
```

```
01 BA 024D 383      popr    #^M<r0>      ; Record return address
08 AE 04 AE 024F 384 10$: addl2  u(sp),y(sp) ; Add the low longwords
OC AE 14 AE 0253 385      adwc    y(sp),z(sp) ; Add the high longwords with the carry
      1F 0258 386      bcs     20$      ; If the result is greater than a quadword
FFFFFFFF 8F 0C AE D1 025A 387      cmpl   z(sp), #-1
      1F 0262 388      blssu   30$      ; or simply greater than or equal to P
FFFFFFFFC5 8F 08 AE D1 0264 389      cmpl   y(sp), #-a
      08 1F 026C 390      blssu   30$      ; we must subtract P.
      08 AE 3B 0C 026E 391 20$: addl2  #a,y(sp)
      OC AE 00 D8 0272 392      adwc    #0,z(sp)
      5E 08 AE 7E 0276 393 30$: movaq   Y(sp),sp ; Point the stack to the return value
      60 17 027A 394      jmp     (r0)      ; return
      027C 395 .dsabl  LSB
      027C 396
      027C 397 .END
```

HPWD  
Symbol table

- hash user password

I 1

16-SEP-1984 02:05:23 VAX/VMS Macro V04-00  
5-SEP-1984 01:45:37 [LOGIN.SRC]HPWD.MAR;1

Page 9  
(4)

A = 0000003B  
AUTODIN 00000000 R 01  
C 00000040 R 01  
COLLAPSE\_R2 00000106 R 01  
EMULQ 00000218 R 01  
ENCRYPT = 0000000C  
LG1\$HPWD 00000068 RG 01  
NO = 00FFFFFFD  
N1 = 00FFFFFFC1  
OUTDSC = 00000004  
PQADD\_R0 0000024D R 01  
PQEXP\_R3 0000017B R 01  
PQLSH\_R0 0000023C R 01  
PQMOD\_R0 000001BB R 01  
PQMUL\_R2 000001D9 R 01  
PURDY 0000011F RG 01  
PWDDSC = 00000008  
SALT = 00000010  
U = 00000000  
USRDSC = 00000014  
V = 00000004  
Y = 00000008  
Z = 0000000C

+-----+  
! Psect synopsis !  
+-----+

| PSECT name | Allocation       | PSECT No. | Attributes |     |     |     |     |       |       |      |       |       |      |  |  |  |  |
|------------|------------------|-----------|------------|-----|-----|-----|-----|-------|-------|------|-------|-------|------|--|--|--|--|
| ABS        | 00000000 ( 0.)   | 00 ( 0.)  | NOPIC      | USR | CON | ABS | LCL | NOSHR | NOEXE | NORD | NOWRT | NOVEC | BYTE |  |  |  |  |
| _LIB\$CODE | 0000027C ( 636.) | 01 ( 1.)  | PIC        | USR | CON | REL | LCL | SHR   | EXE   | RD   | NOWRT | NOVEC | BYTE |  |  |  |  |

+-----+  
! Performance indicators !  
+-----+

| Phase                  | Page faults | CPU Time    | Elapsed Time |
|------------------------|-------------|-------------|--------------|
| Initialization         | 29          | 00:00:00.08 | 00:00:00.33  |
| Command processing     | 107         | 00:00:00.45 | 00:00:02.42  |
| Pass 1                 | 94          | 00:00:00.81 | 00:00:04.61  |
| Symbol table sort      | 0           | 00:00:00.01 | 00:00:00.01  |
| Pass 2                 | 80          | 00:00:00.53 | 00:00:02.62  |
| Symbol table output    | 3           | 00:00:00.02 | 00:00:00.02  |
| Psect synopsis output  | 1           | 00:00:00.01 | 00:00:00.01  |
| Cross-reference output | 0           | 00:00:00.00 | 00:00:00.00  |
| Assembler run totals   | 316         | 00:00:01.91 | 00:00:10.02  |

The working set limit was 1050 pages.  
7793 bytes (16 pages) of virtual memory were used to buffer the intermediate code.  
There were 10 pages of symbol table space allocated to hold 23 non-local and 17 local symbols.  
397 source lines were read in Pass 1, producing 16 object records in Pass 2.  
2 pages of virtual memory were used to define 2 macros.

+-----+  
! Macro library statistics !  
+-----+

| Macro library name                  | Macros defined |
|-------------------------------------|----------------|
| -----                               | -----          |
| \$255\$DUA28:[LOGIN.OBJ]LOGIN.MLB;1 | 0              |
| \$255\$DUA28:[SYS.OBJ]LIB.MLB;1     | 0              |
| \$255\$DUA28:[SYSLIB]STARLET.MLB;2  | 0              |
| TOTALS (all libraries)              | 0              |

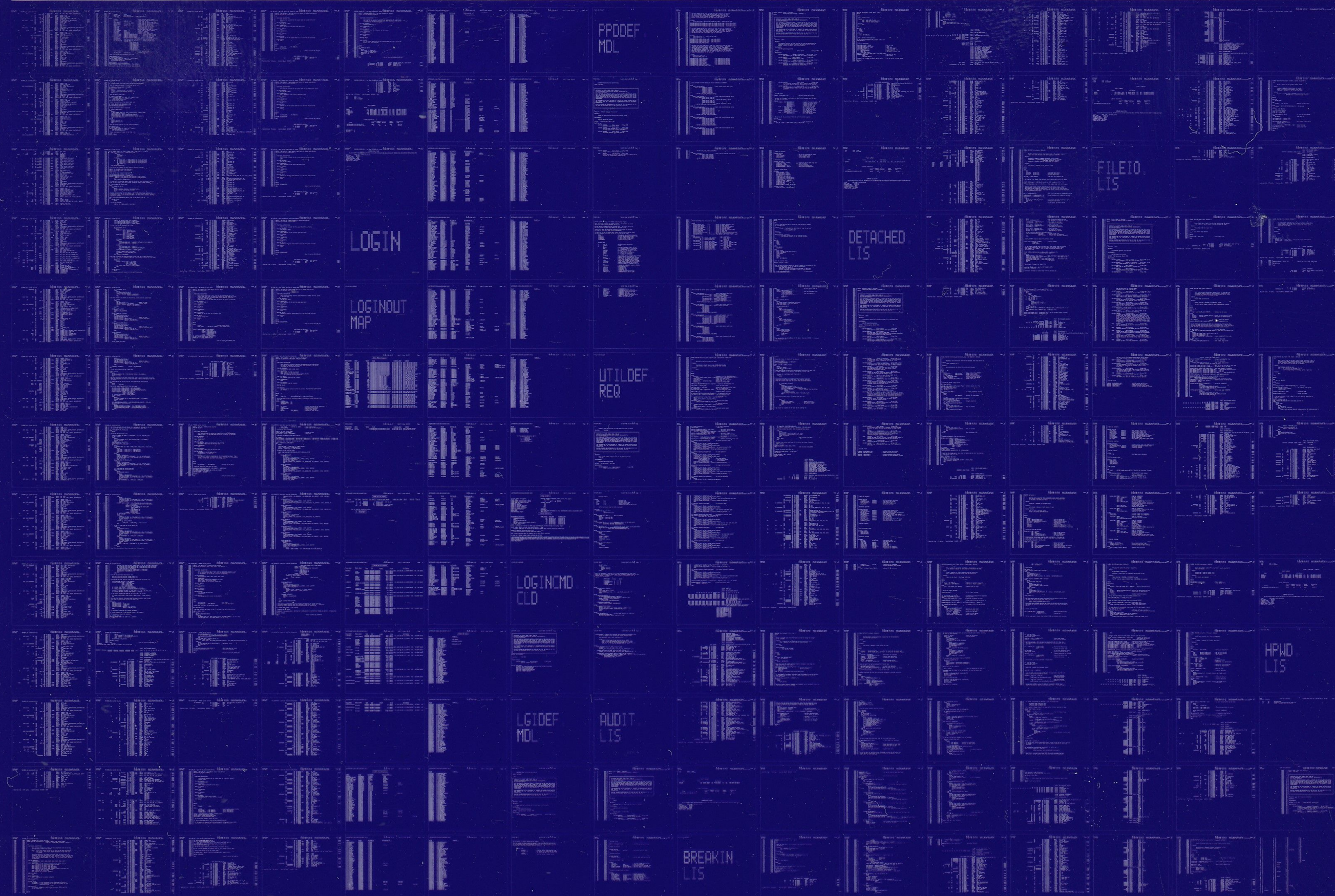
0 GETS were required to define 0 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:HPWD/OBJ=OBJ\$:HPWD MSRC\$:HPWD/UPDATE=(ENH\$:HPWD)+EXECML\$/LIB+LIB\$:LOGIN/LIB

0221 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY



0222 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY

