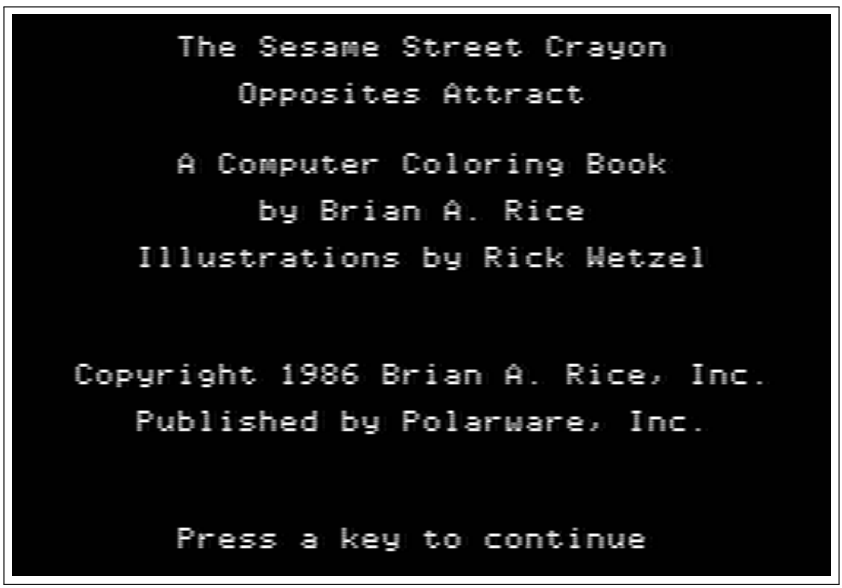
the Sesame Street Crayon
Opposites Attract

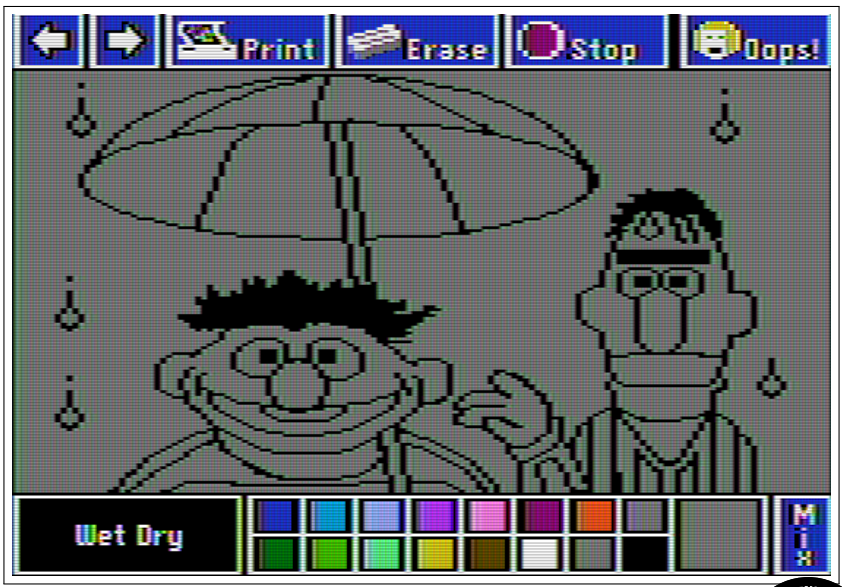
2015-01-20

4am
to deprotect
and preserve

Contents



```
-----------Opposites Attract-----------
A 4am crack                   2015-01-20
---------------------------------------

Name: The Sesame Street Crayon:
  Opposites Attract
Genre: educational
Year: 1986
Publisher: Polarware, Inc.
Media: single-sided 5.25-inch floppy
Authors: Brian A. Rice, Rick Wetzel
OS: ProDOS 1.1.1
Other versions: none (preserved here
  for the first time)
Identical cracks: Grammar Mastery II
  (4am crack no. 189)
```



Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

COPYA
  immediate disk read error

Locksmith Fast Disk Backup
  can't read any track

EDD 4 bit copy (no sync, no count)
  no errors, but copy only boots as far
  as ProDOS title screen, then gives
  "RELOCATION / CONFIGURATION ERROR"

Copy][+ nibble editor
  modified address epilogue "AF AB AB"
  odd-numbered tracks (1, 3, 5...) also
    have a modified address prologue
    ("D4 AA 96")

Disk Fixer
  ["O" -> "Input/Output Control"]
  set Address Epilogue to "AF AB AB"
  -> even-numbered tracks readable
  set Address Prologue to "D4 AA 96"
  -> odd-numbered tracks also readable
  T00 -> looks like ProDOS

Why didn't COPYA work?
  modified prologue/epilogue bytes

Why didn't Locksmith FDB work?
  modified prologue/epilogue bytes

Why didn't my EDD copy work?
  I don't know. The error is a standard
  ProDOS message, but it could easily
  have been triggered manually after a
  failed nibble check.

Converting the disk to a standard format will be tricky. Super Demuffin assumes all tracks share the same prologue and epilogue bytes, but this disk's address prologue alternates between "D5 AA 96" and "D4 AA 96". Advanced Demuffin requires a DOS 3.3-shaped RWTS, but this disk uses ProDOS.

Next steps:

1. Build an RWTS that can read the original disk
2. Convert it to a standard format with Advanced Demuffin
3. Patch the bootloader and/or the PRODOS file to be able to read a standard format disk
4. Find the nibble check (or whatever is triggering the relocation error on the EDD copy) and bypass it

Chapter 1
Bit Math Is Best Math

```
[S6,D1=original disk]
[S6,D2=blank disk]
[S5,D1=my work disk]

]PR#5
CAPTURING BOOT0
...reboots slot 6...
...reboots slot 5...
SAVING BOOT0

]BLOAD BOOT0,A$800
]CALL -151

*801L
.
. standard ProDOS bootloader, until...
.
0831-   85 40       STA   $40
0833-   85 48       STA   $48
0835-   A0 63       LDY   #$63
0837-   B1 48       LDA   ($48),Y
0839-   99 94 09    STA   $0994,Y
083C-   C8          INY
083D-   C0 EB       CPY   #$EB
083F-   D0 F6       BNE   $0837
0841-   A2 06       LDX   #$06
0843-   BC 1D 09    LDY   $091D,X
0846-   BD 24 09    LDA   $0924,X
0849-   99 F2 09    STA   $09F2,Y
084C-   BD 2B 09    LDA   $092B,X
084F-   20 48 09    JSR   $0948    <- !
0852-   CA          DEX
0853-   10 EE       BPL   $0843
```

Standard ProDOS does have this memory copy loop at $0841..$0854, but it does not have any JSR in it. Normally, the instruction at $084F is "STA $0A7F,X", and $0948 is part of the routine that displays the "UNABLE TO LOAD PRODOS" message if something goes wrong during early boot.

```
*9600<C600.C6FFM

; ProDOS boot0 is sensitive to the
; value of the accumulator, so don't
; clobber it
96F8-   48          PHA

; set up callback after copy loop
96F9-   A9 4C       LDA   #$4C
96FB-   8D 55 08    STA   $0855
96FE-   A9 0C       LDA   #$0C
9700-   8D 56 08    STA   $0856
9703-   A9 97       LDA   #$97
9705-   8D 57 08    STA   $0857

; restore accumulator
9708-   68          PLA

; start the boot
9709-   4C 01 08    JMP   $0801
```

```
; callback is here -- save the entire
; bootloader to the hi-res graphics
; page so it will survive a reboot
970C-   A2 03       LDX   #$03
970E-   A0 00       LDY   #$00
9710-   B9 00 08    LDA   $0800,Y
9713-   99 00 28    STA   $2800,Y
9716-   C8          INY
9717-   D0 F7       BNE   $9710
9719-   EE 12 97    INC   $9712
971C-   EE 15 97    INC   $9715
971F-   CA          DEX
9720-   D0 EE       BNE   $9710

; turn off the slot 6 drive motor
9722-   AD E8 C0    LDA   $C0E8

; reboot to my work disk
9725-   4C 00 C5    JMP   $C500

*BSAVE TRACE,A$9600,L$128
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1 0800-0AFF,A$2800,L$300
]BLOAD BOOT1 0800-0AFF,A$800
]CALL -151

*93FL

; this is the start of the routine that
; normally displays the "UNABLE TO LOAD
; PRODOS" message, but it's been
; shortened to just call $FF2D (beeps
; and prints "ERR") instead
093F-   20 58 FC    JSR   $FC58
0942-   20 2D FF    JSR   $FF2D
0945-   4C 45 09    JMP   $0945
```

```
; this is the subroutine called from
; the copy loop at $084F, and the first
; instruction here is the one that was
; clobbered by the call to this
; subroutine
0948-   9D 7F 0A     STA   $0A7F,X
094B-   BD 5C 09     LDA   $095C,X
094E-   9D F7 09     STA   $09F7,X
0951-   BD 63 09     LDA   $0963,X
0954-   9D FE 09     STA   $09FE,X
0957-   A9 AA        LDA   #$AA
0959-   85 31        STA   $31
095B-   60           RTS
```

ProDOS normally boots by copying part of the drive controller ROM routine (at $C65C or wherever, depending on the boot slot) into RAM and massaging it to create a working RWTS. This is how it can fit an entire bootloader in three pages -- the hard part of reading the disk is already taken care of.

But on this disk, there is some extra massaging. For example, this snippet gets dropped into the middle of the RWTS code:

```
095C-   4A           LSR
095D-   C9 6A        CMP   #$6A
095F-   D0 F3        BNE   $0954
```

And this one:

```
0961-   BD 8C C0     LDA   $C08C,X
0964-   10 FB        BPL   $0961
0966-   C9 AA        CMP   #$AA
0968-   D0 EA        BNE   $0954
```

The upshot is that the final RWTS is different than the drive controller ROM routine. After all the memory massaging is complete, this is the part of the constructed RWTS that checks for the address and data prologue:

```
*9EFL

09EF-   88          DEY
09F0-   F0 F5       BEQ   $09E7

; prologue nibble #1
09F2-   BD 8C C0    LDA   $C08C,X
09F5-   10 FB       BPL   $09F2
09F7-   4A          LSR             |dif
09F8-   C9 6A       CMP   #$6A      |fer
09FA-   D0 F3       BNE   $09EF     |ent

; #2
09FC-   BD 8C C0    LDA   $C08C,X
09FF-   10 FB       BPL   $09FC
0A01-   C9 AA       CMP   #$AA
0A03-   D0 EA       BNE   $09EF

; #3
0A05-   BD 8C C0    LDA   $C08C,X
0A08-   10 FB       BPL   $0A05
0A0A-   C9 96       CMP   #$96
0A0C-   F0 09       BEQ   $0A17
```

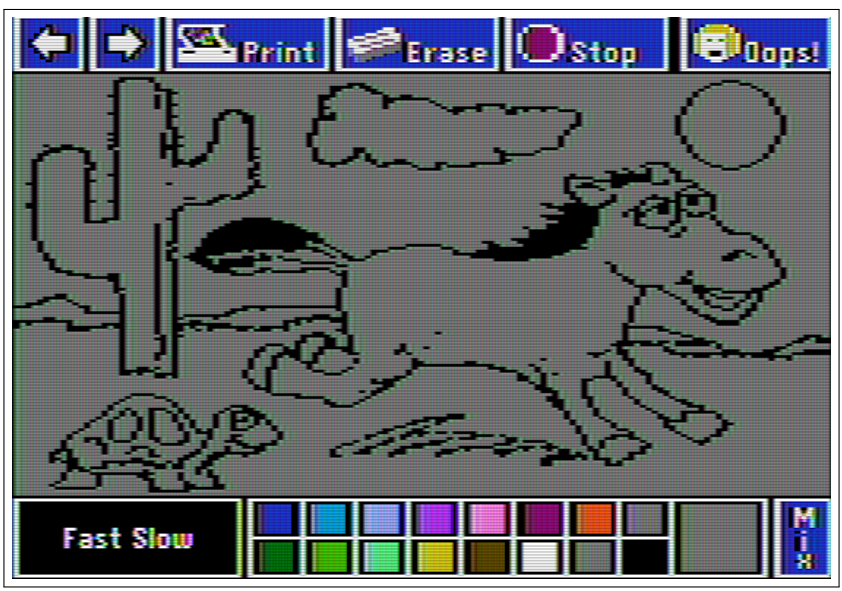
The code to find prologue nibble #1 explains how this disk can read its odd-numbered tracks (with non-standard address prologue "D4 AA 96").

```
Normal address prologue byte 1 is $D5.
In binary: $D5 = 1101 0101
After LSR:      0110 1010 = $6A

Odd-numbered tracks use $D4 instead.
In binary: $D4 = 1101 0100
After LSR:      0110 1010 = $6A

So this code will match either prologue
and work on both odd and even tracks.
Clever!

I can use this same technique to build
a flexible DOS 3.3-shaped RWTS that can
read the even- and odd-numbered tracks.

[S6,D1=DOS 3.3 master disk]
[S5,D1=my work disk]

]PR#6
]CALL -151

*2800<B800.BFFFM

*294FL

294F-   BD 8C C0    LDA   $C08C,X
2952-   10 FB       BPL   $294F
2954-   C9 D5       CMP   #$D5
2956-   D0 F0       BNE   $2948
2958-   EA          NOP

*2954:4A C9 6A D0 EF
```

```
*294FL

294F-    BD 8C C0     LDA    $C08C,X
2952-    10 FB        BPL    $294F
2954-    4A           LSR
2955-    C9 6A        CMP    #$6A
2957-    D0 EF        BNE    $2948

; set custom address epilogue bytes
*2991:AF
*299A:AB

*BSAVE RWTS LSR 6A,A$2800,L$800,S5,D1
```

Chapter 2
In Which We Normalize The Universe, Or Failing That, At Least This Disk

```
[S6,D1=original disk]
[S6,D2=blank disk]
[S5,D1=my work disk]

]PR#5
]BRUN ADVANCED DEMUFFIN 1.5

["5" to switch to slot 5]

["R" to load a new RWTS module]
   --> At $B8, load "RWTS LSR 6A" from
       drive 1

["6" to switch to slot 6]

["C" to convert disk]
```



```
                    --v--

ADVANCED DEMUFFIN 1.5    (C) 1983, 2014
ORIGINAL BY THE STACK    UPDATES BY 4AM
=======PRESS ANY KEY TO CONTINUE=======
TRK:.....................................
+.5:
    0123456789ABCDEF0123456789ABCDEF012
SC0:.....................................
SC1:.....................................
SC2:.....................................
SC3:.....................................
SC4:.....................................
SC5:.....................................
SC6:.....................................
SC7:.....................................
SC8:.....................................
SC9:.....................................
SCA:.....................................
SCB:.....................................
SCC:.....................................
SCD:.....................................
SCE:.....................................
SCF:.....................................
=========================================
16SC $00,$00-$22,$0F BY1.0 S6,D1->S6,D2

                    --^--

]PR#6
...program boots and runs...

Wait, what?
```

Chapter 3
In Which We Angrily Investigate Why We Suddenly Have A Working Copy

```
[S6,D1=mysteriously working copy]
[S7,D1=ProDOS hard drive]

]PR#7
]CAT,S6,D1

/OPPOSITES

 NAME              TYPE  BLOCKS  MODIFIED

*PRODOS            SYS       30  18-SEP-84
 EC.SYSTEM         SYS       13  12-AUG-87
 EC.MAIN.OBJ       BIN       13  12-AUG-87
 NOMENUS.TK.ABS    BIN       23  13-DEC-85
 TEST.FONT         BIN        6   7-NOV-85
 EC.COLOR.DRIVER   BIN       10  16-DEC-85
 EC.IMAGE2.SETUP   BIN        1   9-DEC-85
 EC.SCRIBE.SETUP   BIN        1   9-DEC-85
 EC.ADSWITCH.OBJ   BIN        1   9-DEC-85
 EC.JDRIVER.OBJ    BIN        3   9-DEC-85
 EC.KDRIVER.OBJ    BIN        4   9-DEC-85
 EC.SCREEN.CLIP    BIN        9  12-DEC-85
 EC.PARMS          BIN        1   8-APR-87
 EC.COLORS         BIN       17  11-MAR-87
 EC.DIALOG.OBJ     BIN        4  12-AUG-87
 SHAPES            BIN        1  19-AUG-86
*PICTURES          DIR        3  <NO DATE>

BLOCKS FREE:    29      BLOCKS USED:  251
```

```
]PREFIX /OPPOSITES
]BLOAD PRODOS,A$2000,TSYS
]CALL -151

; ProDOS only uses the bootloader RWTS
; to load the PRODOS file, which then
; has its own fuller, more robust RWTS.
; This code, which is later relocated
; to $D398 in the language card, checks
; the address prologue.
5398-   A0 FC       LDY   #$FC
539A-   8C 6B D3    STY   $D36B
539D-   C8          INY
539E-   D0 05       BNE   $53A5
53A0-   EE 6B D3    INC   $D36B
53A3-   F0 56       BEQ   $53FB

; find prologue byte #1
; (matches $D4 or $D5)
53A5-   BD 8C C0    LDA   $C08C,X
53A8-   10 FB       BPL   $53A5
53AA-   4A          LSR
53AB-   C9 6A       CMP   #$6A
53AD-   D0 EE       BNE   $539D

; #2
; (zero page $31 was initialized during
; boot to $AA)
53AF-   BD 8C C0    LDA   $C08C,X
53B2-   10 FB       BPL   $53AF
53B4-   C5 31       CMP   $31
53B6-   D0 F2       BNE   $53AA
53B8-   A0 03       LDY   #$03
```

```
; #3
53BA-   BD 8C C0    LDA   $C08C,X
53BD-   10 FB       BPL   $53BA
53BF-   C9 96       CMP   #$96
53C1-   D0 E7       BNE   $53AA
```

No surprises here. All tracks on the converted disk use "D5 AA 96", which always matches, so it always finds the address prologue.

But look at the address epilogue check a few lines further down:

```
*53E6L

; find epilogue byte #1
53E6-   BD 8C C0    LDA   $C08C,X
53E9-   10 FB       BPL   $53E6
53EB-   C9 DE       CMP   #$DE

; if found $DE, immediately exit with
; a "success" status (clear carry bit)
53ED-   F0 0A       BEQ   $53F9

; if not $DE, do... this thing
53EF-   48          PHA
53F0-   68          PLA
53F1-   BD 8C C0    LDA   $C08C,X

; Note: no BPL loop here! It only reads
; the data latch once.
53F4-   C9 08       CMP   #$08
53F6-   B0 03       BCS   $53FB
53F8-   EA          NOP
53F9-   18          CLC
53FA-   60          RTS
53FB-   38          SEC
53FC-   60          RTS
```

It's looking for a timing bit after the first epilogue byte. It doesn't even care what the first epilogue byte was, as long as it wasn't $DE.

This RWTS will accept two different address prologues, "D5 AA 96" or "D4 AA 96". It will also accept two different address epilogues, "DE" or anything-other-than-DE-followed-by-a-timing-bit.

Why didn't the EDD copy work?
  The bootloader RWTS doesn't check epilogue bytes at all, so it was able to read the disk and load the PRODOS file. Once control is transferred to the PRODOS file, it switches to its own RWTS to read the disk catalog and find the first .SYSTEM file. But its own RWTS can't read the disk, because EDD preserved the original prologue epilogue but not the timing bits. The prologue checker (at $D398) finds "D5 AA 96" even-numbered tracks) or "D4 AA 96" (odd-numbered tracks). But the epilogue checker's first compare (at $D3EB) didn't match because the first epilogue byte was still the original value ($AF), and its second compare (at $D3F4) didn't match because there was no timing bit after the first byte. ProDOS can't read the disk catalog, so it displays the "RELOCATION / CONFIGURATION ERROR" and gives up. There was never any nibble check; the very structure of the disk itself is designed to foil bit copiers.

Why did the demuffin'd copy work?
  Advanced Demuffin wrote out the data from each sector onto a standard disk that uses "D5 AA 96" prologue and "DE AA EB" epilogue. The bootloader RWTS always matches "D5 AA 96" and doesn't care that it never sees a "D4 AA 96", and it never checks epilogue bytes at all. The RWTS within the PRODOS file always matches "D5 AA 96", and its epilogue checker always matches "DE" and never checks the timing bit. Thus no RWTS patches are necessary.

Quod erat liberandum.

EOF