My very first self-written encryption program (~1980)

Bernd Fix <brf@hoi-polloi.org>

Introduction

During my last year at school (around 1980) the new-founded “computer lab” at our school was equipped with some four Commodore PET 8032 computers, which captured my interest early on and allowed me to expand my computer expertise that started when working on an older PET 2001 two years before.

The main advantage of the 8032 boxes was the availabillity of a floppy drive (although the 8032 was still equipped with an external datasette), but floppies where expansive at that time and could not be bought anywhere close to the place I lived. Luckily the “lab” provided a diskette for every user, but you were not allowed to take these home – they where kept in an (unlocked) box at school. So anyone could get the floppy of another user and have a look at the programs (and data) stored on them.

All my assembler programs were quite “safe” – just one or two other users were able to read them anyway and these were kind of friends, so I didn't worry about that. But all the BASIC programs I wrote were “open” to everyone – especially the teachers. So I thought of a way to protect these BASIC programs from unwanted inspections...

I wrote a small assembler program called “CIPHER” to en- and decrypt these files. I stored my BASIC programs encrypted on the floppy and decrypted them before editing or executing them. The cipher application itself was stored on an audio tape (to be used with the datasette), so I could take it home with no problems.

After leaving school and going to university I worked on quite a few more interesting computer systems (UNIVAC 1108, PDP-11, IBM/370), so I forgot about the PET 8032 – it simply wasn't “sexy” anymore. But for whatever reason I kept the audio tape with the cipher program...

After some 30 years I re-discovered that audio tape – my very first self-written encryption program! Since my main interest in computers since the mid-80's was computer security (and therefore cryptology), I decided to re-engineer it – there was no documentation whatsoever, just the plain binary file on a tape. Using VICE on my Linux box I actually got it to run...

Loading the CIPHER program

The program on tape was somewhat protected from being loaded by someone not too familiar with the system – the name contained characters that prevented it from being fetched using a simple LOAD command. You had to do something like the following to get it loaded:

```
s$=chr$(34)+"cipher"+chr$(34):load s$,1
press play on tape #1
ok
searching for "cipher"
found "cipher"
loading
ready.
```

The assembler program was loaded into the memory from $033A to $03FF, which is used as the buffer for a second datasette drive. Since no second datasette was attached to the system, the program stayed there without being destroyed by an other (BASIC) program on the computer – and it was resident until the computer was resetted or turned off.

Encrypting a BASIC program

BASIC programs on the PET are a linked list of memory chunks that each represent a single (numbered) line of the program, so editing a program usually fragments the memory. To get it packed after editing, you have to store it in clear-text form first – it will later be overwritten by the encrypted program.

Lets assume our program is stored on the floppy under the name TEST. First we have to find the size of the program in memory by getting the free memory space before and after loading the program:

```
new
ready.
print fre(0)
 31741
ready.
load "test",8
searching for test
loading
ready.
list
 10 for i=1 to 100
 20 print i
 30 next i
ready.
print fre(0)
 31709
ready.
```

The simple TEST program occupies 32 bytes (31741-31709), but we have to add some bytes to cover leading and trailing bytes. I used to add 8 bytes for that (just to be save – three bytes would be sufficient), so the total length in our example is 40 bytes ($28).

The next step is to invoke the CIPHER program via the SYS command; we are asked for a code (passphrase) that is terminated by the return key. After encryption a LIST command shows only garbage:

```
sys 908
code :
testkey
program de-/ciphered
ready.
list
 21155 $+restorecmd%sysson:director
                                    +stepexpagp.!formula too
complex+.<cos
 43434 expagp.!formula too complex+.<cos
 59914 +.<cos
ready.
```

You can't save the encrypted program with the standard SAVE command – I guess the SAVE routine tries to “interpret” the program and get messed up. You have to use the built-in monitor (TIM) to save it and that requires the knowledge of the memory range to be saved. BASIC programs always start at address $0400 and the end address can be calculated from the program length we have determined before:

```
sys 1024
b*
     pc  irq  sr ac xr yr sp
.;  0401 e455 32 04 5e 00 f8
.s "test",#8,0400,0428
.x
ready
```

Now the encrypted BASIC program is stored on the floppy.

Decrypting a BASIC program

You can't load an encrypted program with the standard LOAD command for probably the same reason why you can't save it with the standard SAVE command. Again we have to use TIM to read it back and start the decrpytion. After that the LIST program shows the original and executable program again:

```
sys 1024
b*
     pc  irq  sr ac xr yr sp
.;  0401 e455 32 04 5e 00 f8
.l "test",#8
searching for test
loading
.g 038c
code :
testkey
program de-/ciphered
?syntax error
ready.
list
 10 for i=1 to 100
 20 print i
 30 next i
ready.
```

The CIPHER program

To compile the CIPHER program I used an assembler that was written by a friend and myself in BASIC – that program now resides in the digital nirvana and is lost forever. So no source code for the CIPHER in its original form is available; the following is a manual disassembly of the binary executable that survived the fimes. If you want to recompile it with an assembler, you probably have to make some minor adjustments to the code.

One mystery remains: the last JMP instruction branches to a ROM routine and I have no idea what that really does – I simply can't remember. Probably it does some clean-up, breaks from TIM or whatever – running it from TIM under VICE results in an “?syntax error” that was probably not the case in the original version. Maybe I am just using a different ROM image with VICE, so its not doing what was intended. But it seems to work anyhow...

```
           *=$033A
;-----------------------------------------
; Cipher stream generator: 4-byte LFSR in
; zero page registers (16-19); cipher byte
; is lowest register (16).
; Feedback bit is (bit31 ^ bit7).
;-----------------------------------------
CSG        TXA
           PHA
           LDX #$07
L1         LDA $13
           EOR $10
           ASL
           ASL
           ROL $10
           ROL $11
           ROL $12
           ROL $13
           DEX
           BPL L1
```

```
           PLA
           TAX
           RTS
;-----------------------------------------
; En-/decrypt BASIC program (line by line)
; - init pointer (1,2) to 0x401 (first line)
; - save address of next line in (3,4)
;   terminate if next address is zero
; - compute length of line in X
; - process next byte in line
; - set address of next line as current
;-----------------------------------------
PROC       LDA #$01
           STA $1
           LDA #$04
           STA $2
L5         LDY #$00
           LDA ($01),Y
           STA $3
           INY
           LDA ($01),Y
           BNE L2
           RTS
L2         STA $04
           SEC
           LDA $03
           SBC $01
           BCS L3
           ADC $FF
L3         TAX
           DEX
           DEX
           DEX
L4         INY
           JSR CSG
           LDA ($01),Y
           EOR $10
           STA ($01),Y
           DEX
           BNE L4
           LDA $03
           STA $01
           LDA $04
           STA $02
           BNE L5
;-----------------------------------------
; Program entry point ('SYS 908', $038C)
; - prompt for code sequence
; - store code input in buffer
; - clear/reset LFSR
; - set LFSR from code input (length in X)
; - en-/decrypt BASIC program
; - print success message
; - clean-up? break from TIM?
;-----------------------------------------
ENTRY      LDX #$00
L6         LDA MSG1,X
           JSR $FFD2
           INX
           CPX #$0A
           BNE L6
           LDX #$00
L8         JSR $FFCF
           CMP #$0D
           BEQ L7
           STA ($20),X
           INX
           BNE L8
L7         LDY #$03
```

```
            LDA #$00
L9          STA $10,Y
            DEY
            BPL L9
            STX $00
            LDX #$00
L12         LDY #$03
L11         LDA $20,X
            EOR $10,Y
            STA $10,Y
            INX
            CPX $00
            BMI L10
            DEY
            BPL L11
            BMI L12
L10         JSR PROC
            LDX #$00
L13         LDA MSG2,X
            JSR $FFD2
            INX
            CPX #$18
            BNE L13
            JMP $C735
;-----------------------------------------
; Application messages
;-----------------------------------------
            .byte $EA,$EA
MSG2        .byte $0D,$0D,$50,$52,$4F,$47,$52,$41   ; ..PROGRA
            .byte $4D,$20,$44,$45,$2D,$2F,$43,$49   ; M DE-/CI
            .byte $50,$48,$45,$52,$45,$44,$0D,$0D   ; PHERED..
MSG1        .byte $93,$0D,$43,$4F,$44,$45,$20,$3A   ; ..CODE :
            .byte $0D,$0D
```

Final thoughts

The encryption is – using todays and even yesterdays standards – far from secure in any way, but it was at that time certainly sufficient to keep teachers from inspecting my programs. It uses only a 31 bit key, so it will nowadays take just seconds to break the cipher. On the other hand, there are certainly not many other encryption programs around that do a better job with just 198 bytes of program code.