ZX Vega Firmware

As I only have the compiled software, this is slightly tricky, however there are many binary tools available to aid with initial tests.

Most of the output would be garbage, so some patient scrolling is required! However, there are some snippets of helpful information to get us started.

```
VEGA@
0.1.64 20150909
The Sinclair ZX Spectrum Vega.
Copyright (C) 2015 Retro Computers Limited.
www.retro-computers.co.uk  www.zxvega.co.uk
Concept, CAD, hardware design, C, ARM and Z80
programming: Chris Smith. www.zxdesign.info
SMEM
The Sinclair ZX Spectrum Vega.
Copyright (C) 2015 Retro Computers Limited.
www.retro-computers.co.uk  www.zxvega.co.uk
Concept, CAD, hardware design, C, ARM and Z80
programming: Chris Smith. www.zxdesign.info
STMP
2m?`
kvjU<v
```

This is surprisingly helpful! We can see the copyright notices, but more importantly three chunk headers (VEGA, SMEM, STMP)

Later on, we have this:-

```
mkdosfs
NO NAME    FAT16
This is not a bootable disk.  Please insert a bootable floppy and
press any key to try again ...
```

Again, this is handy as we know the filesystem is FAT16 made by mkdosfs. This will help us later on.

Ignoring the first two chunked headers, we are interested in the third – STMP, as this tells us where the file format comes from, which will help us proceed. A quick look on google gives us https://www.rockbox.org/wiki/SbFileFormat

there is also a copy of the tool at https://github.com/eewiki/elftosb
And, as we know the main chip is from NXP (renamed from freescale), there is further

documentation at: https://www.nxp.com/docs/en/user-guide/KBLELFTOSBUG.pdf

Using the documents from Rockbox, we can now decode the STMP chunk.

| Start | Length | Type | Value | Use |
|---|---|---|---|---|
| 0x00 | 20 | uchar | 35c5b7bf564c0c6deca694c5b48 071976c03b586 | SHA1(STMP chunk) Also IV for encryption |
| 0x14 | 4 | u32 | “STMP” | Magic |
| 0x18 | 1 | u8 | 0x01 | Major version of format |
| 0x19 | 1 | u8 | 0x01 | Minor version of format |
| 0x1A | 2 | u16 | 0x0000 | Flags |
| 0x1C | 4 | u32 | 0x00008840 | Image size in blocks (0x8840 x 16 = 0x88400) |
| 0x20 | 4 | u32 | 0x00009000 | Offset to first boot tag in blocks (0x9000 x 16 = 0x90000) |
| 0x24 | 4 | u32 | 0x00000000 | First bootable section |
| 0x28 | 2 | u16 | 0x0001 | Number of encryption keys |
| 0x2A | 2 | u16 | 0x0007 | Start block for key dictionary (7 x 16 = 0x70) |
| 0x2C | 2 | u16 | 0x0006 | Size of header in blocks (6 x 16 = 0x60) |
| 0x2E | 2 | u16 | 0x0001 | Number of sections headers |
| 0x30 | 2 | u16 | 0x0001 | Size of chunk headers in blocks |
| 0x32 | 2 | uchar | 0x6136 | Padding |
| 0x34 | 4 | u32 | 0x2d800a09 | Second signature |
| 0x38 | 8 | u64 | 0x0001c256e2a2b280 | Creation time in µs since 2000 |
| 0x40 | 4 | u32 | 0x00000999 | Product major version |
| 0x44 | 4 | u32 | 0x00000999 | Product minor version |
| 0x48 | 4 | u32 | 0x00000999 | Product sub version |
| 0x4C | 4 | u32 | 0x00000999 | Component major version |
| 0x50 | 4 | u32 | 0x00000999 | Component minor version |
| 0x54 | 4 | u32 | 0x00000999 | Component sub version |
| 0x58 | 2 | u16 | 0x0000 | Drive tag |
| 0x5A | 6 | uchar | 0xcd6212000ec5 | Padding |

Following this block we have the DEK block. This is a block of Data Encryption Keys. Each of these is created with a Key Encryption Key (KEK). This is where we end. We don't have the KEK.

According to the rockbox documentation, the key uses AES-128 in CBC mode to encrypt the DEK with a KEK. In non-technical terms, this means there are millions of different combinations needed to brute force the key.

Not to say we can't use side channel mechanisms instead...