```
     ___        /  /\
    /__/\      /  /::\
    \__\:\    /  /:/\:\
    /  /::\  /  /:/  \:\
 __/  /:/\/ /__/:/ \__\:\
/__/\/:/~~  \  \:\ /  /:/
\  \::/      \  \:\  /:/
 \  \:\       \  \:\/:/
  \__\/        \  \::/
                \__\/
```

# Intel ME (Manageability engine) Huffman algorithm

Starting at version 6 the "firmware" for the Intel manageability engine (sometimes: management engine) uses a custom compression scheme. Some of the modules are compressed with standard lzma, but others use a custom scheme whose details remained unknown until this publication. Making it impossible to inspect and audit modules compressed with it. The ME runs its own operating system on a dedicated seperate cpu inside all modern intel chipsets. With direct access to hardware, including all memory, and networking(including wifi). Is capable of running while the system is in S3 (aka off). It has code to break into your graphics/input devices outside of the Host OS' control (for secure PIN input). It provides pavp (protected audio video path) also known as DRM. It can be configured as the ultimate RAT. Code running on the ME cpu is often said to be running in ring -3. In short there are plenty of reasons to be curious.

**Scheme details:**

- the chipset has 2 dictionaries, stored in hardware only
- in ME version 6 one of them is used to compress code, the other data
- higher ME versions this difference isn't clear.
- dictionaries consist out of a large set of prefix free codes
- prefix free codes are 7 to 19 bits long
- each prefix free code translates into 1 to 15 bytes
- compression is poor, way worse than gzip -1, but fast decompression
- for v6 compressed file is 77.5% the size of the original on average
- dictionaries are not present in firmware
- prefix free codes form a canonical tree
- depending on the version a single dictionary contains about 1700-2100 entries
- often multiple ways to encode plaintext, the shortest possibility is _almost_ always used.
- redundant symbols exist. (one that can be expressed by multiple other symbols equally or more efficiently)

**Obtaining firmware**
The tool presented here unpacks the ME code stored on SPI flash that the ARCompact cpu executes. To get an image of this code you can:

- extract it from bios updates. (It is often shipped with bios update images. binwalk, custom, ..)
- use a hardware SPI programmer
- use flashrom from flashrom.org

**Does it affect me?**

If you are using an intel system that's less than 5 years old, almost definitely. These systems can't boot without the ME. The fsf certified glugglug laptop is probably the only system you can buy that disables the ME.

Tablets and other devices based on an Atom SoC don't have a traditional chipset and no

ME. Instead they have a "security engine" which basically does the same, but it is a different archititecture (SPARC) and does not yet use the custom compression scheme.

What capabilities are enabled, however differ greatly from OEM to OEM. You can use this tool by Damien Zammit to query the status of the ME on your system. It uses the MEI (ME interface, previously called HECI) a bidirectional PCI interface between the host cpu and the ME cpu.

**Missing code, storage format**

As mentioned in several publications by Igor Skochinsky: the chipset has some inaccesible ROM. Besides likely storing the decompression dictionaries, it also harbours a code library. The library implements libc-style primitives (memcpy, ...) and who knows what else. The code unpacked by unhuffme will likely contain function calls into this library via what is referred to as RAPI (rom api). Despite the ROM being inaccessible, the purpose of these calls can typically be deduced by reversing, or by comparing with a version of the code which bypasses ROM. However this library can still not be audited directly.

File format information directly based on Igor Skochinsky's research. His me_unpack supports more versions and features of the firmware, but does not decompres the huffman compressed sections.

**Unhuffme v2.4**

Unpack ME versions 6 through 10
 Linux Source: unhuffme-v2.4.tar.xz
 Windows CLI build: unhuffme-v2.4.1.zip

**Thanks**

```
Indispensable help by:
        Igor Skochinsky
        phcoder
        Corey Kallenberg
        Xeno Kovah
        Rafal Wojtczuk
```

**Completeness**

Unhuffme v2.4 completely unpacks all modules from all observed ME images, including SPS ME 3.x.

When unpacking version 6 firmwares you will find that the module "VE_FW_NAND" fails consistently. Rather than not being able to decompress it, the compressed data is missing.

If you come across any other module not unpacking cleanly, please provide a copy.

The chipsets associated with Skylake microarchitecture released around August 2015 and onwards have a new version of the ME(version 11), with new compression dictionaries which are not public.

--bla <blapost@gmail.com>

Published 8 April 2015 (v2.3)
Published 6 May 2015 (v2.4)
old