CRYPTOGRAPHY OF DECODER

Cryptography

Recall that the basic idea is to build a gadget which takes a 32-bit encrypted number and decrypts it, treating the 32-bit plaintext as two 16-bit milliminute offsets.

It's rather pretentious to call this cryptography, after all what we really want is just a function which scrambles 32-bit quantities parameterized by a secret key. Although it's probably overkill here, it would be nice if the scrambler had the usual good-crypto features:

- small changes in the input should give big changes in the output;
- it should be hard to infer the key from a small number of plain/crypt pairs.

skipjack32

I'm no cryptography expert so it would be daft to invent a scrambling algorithm from scratch: instead it's both quicker and more sensible to look online.

There aren’t a vast number of choices, presumably because 32-bits is a small enough space to succumb to brute-force attacks. Most online examples appear to use a 32-bit varient of skipjack.

Initially I found a Perl implementation which happily includes the original C source written by Greg Rose. Qualcomm’s opensource site appears to host the original. Despite being written in 1999, it still compiles happily today, and I used it in the microcontroller firmware.

For actually working out which encrypted code corresponds to particular coordinates, I reimplemented the algorithm in Haskell.

The Haskell assumes:

```
stdKey =  Key [ 0x00,0x99,0x88,0x77,0x66,0x55,0x44,0x33,0x22,0x11 ]

-- Origin: N 51 12.345 E 0 9.876

stdOffs = (51 * 60000 + 12345, 9876)
```

Given these we can encode coordinates thus:

```
$ ghci skip32.hs

GHCi, version 7.4.2: http://www.haskell.org/ghc/  :? for help

...

*Skip32> putStrLn $ stdEncCheck (52, 12.345) (0, 56.789)
```

```
Crypt: 0xf45655de
Plain: 0xea60b741
  0x002fcbb9 => N 52 12.345
  0x0000ddd5 => E 0 56.789
```

So N 52 12.345 E 0 56.789 is represented by the hex code 0xf45655de. Being paranoid, the code above then decodes that code and checks we get back what we expected. It’s also useful to see the coordinate offset in hex:

```
*Skip32> putStrLn $ stdHexOffsets
  0x002ee159 => N 51 12.345
  0x00002694 => E 0 9.876
```

Source: http://www.mjoldfield.com/atelier/2012/12/coord-decoder.html