lintblog

random projects
18
Aug
Magnetic Stripe ABA Track 2 Encoding
stored in: Magnetic Stripes and tagged: aba, magstripe, track2

This post will cover the encoding scheme defined by the ABA for magnetic stripe cards. This format is the de facto standard for track 2 of magnetic stripe cards.

Character Encoding Overview

- Each character is five bits in length. Four bits for the character itself, and an odd parity bit.
- The character set includes 16 ASCII symbols, ranging from 0×30 – 0x3f on the ASCII table. These are numerical digits 0-9 and the characters : ; < = > ?
- Encoded characters are offset from ASCII by decimal 48 (thus, the numeric digit 0 is stored as 0, and a value of 48 must be added to get the ASCII character 0).
- Values are stored in binary in Least Significant Bit (LSB) order. E.g., the number 4 is stored as 001 and not 100.

I'll go over a few quick examples of this.

The start sentinel is the semi-colon character. This is ASCII value 59, but we subtract 48 from this. We then have decimal 11, and in LSB binary: **1101**. We then apply the parity bit to the end, which is **0** here as there is an odd number of 1s. Thus, the encoded character is **11010**.

Another example, the number 5. This is ASCII 53, but becomes the number 5 once 48 is subtracted. In LSB binary, it is **1010**. There is an even number of 1s, so the parity bit is **1**. The encoded character is **10101**.

Data Formatting

- Always starts with a semi-colon, the “start sentinel”, to indicate beginning of data.
- Always ends with a question mark, the “end sentinel”, to indicate end of data.
- The equal sign is used as a field separator. Traditionally this is put between an account number an expiration date. Some cards will contain no field separators, others may contain multiple.
- Only numeric digits should be used to store data — the remaining characters, : < >, are for hardware control purposes.
- A 5 bit Longitudinal Redundancy Check (LRC) character always follows the end sentinel. It uses an even parity bit scheme, as opposed to the odd parity used for characters. Its fifth bit is an odd parity of the other 4 bits, and not an LRC parity bit.
- A total of 40 characters can be stored, including start/end sentinels and field separators.

Longitudinal Redundancy Calculation

The LRC is calculated by looking at all the other encoded characters on the card and calculating an **even** parity bit.

I’ll go over an example here. Let’s assume we have the string ;12=34? as data to encode:

```
11010  ;
10000  1
01000  2
10110  =
11001  3
00100  4
11111  ?
10110  LRC
```

Just do addition down each column. If the total is odd, then the LRC parity bit for that position is 1. If the total is even, it’s a 0. The fifth bit of the LRC is an **odd** parity of the first four bits. In this case, the LRC is **1011**, which is an odd number of bits, making the encoded LRC **10110**. The fully encoded stream is then:

```
1101010000010001011011001001001111110110
```

Next I'll do a quick overview of how data is commonly ordered on a card:

Common Data Ordering

1. Start Sentinel
2. Primary Account Number (PAN). Up to 19 digits in length. Overflow can be stored in the custom data section.
3. Field Separator
4. Expiration date in the format YYMM
5. Custom data. Length is up to the remaining space.
6. End Separator

This format is found on many cards, such as credit cards. Odds are if you have a magnetic stripe card with an identification number and expiration date, it follows this pattern.

Not all cards will follow this, refer to the data formatting section above for some rules that should be consistently followed.

2 Comments Me gusta

2 Responses to "Magnetic Stripe ABA Track 2 Encoding"

1. *David Chappelle* Says:
   September 22nd, 2010 at 13:02

   Hi, I think this example LRC calculation may not be correct. Based on a number of test cards I've tried, the LRC does not use the start or end sentinels. So in this example, the resulting LRC would be 0110 plus parity bit.

2. *ieatlint* Says:
   September 27th, 2010 at 23:50

   @David Chapelle

   The start and end sentinel are part of the LRC.

   You're suggesting that the LRC is 01101, making the bits:
   11010100000100010110110010010011111101101

This stream is invalid — it fails the LRC when decoding is attempted.

It is possible to not include the start and end sentinel in the loop to calculate the LRC, if you take into account the constant value of the start and end sentinel.
To do it that way seems more complicated to me, but it can be done by taking the LRC as you calculated it, and then doing an even parity bit of the start and end sentinel (0010) and then an odd parity bit of your calculated amount with that.
The process is:

1101 Start sentinel
1111 End sentinel
0010 Even parity of them

0010 The parity of the sentinels
0110 Your calculated LRC
1011 Odd parity of them

plus an odd parity bit of 0 … 10110.

I’ve seen some code that does it this way — and my original revision of mslib did something a bit janky related to this. You can search for that in the git history on github if you’re so inclined.

Anyway, the LRC calculation method as explained in my original post works. It is more straightforward than trying to pick parts of the stream to include, and overall pretty simple. This is the basis of the code in mslib, which reliably works to decode magstripes.

Search Blog

- **Pages**
  - About